DOCUMENT RESUME

ED 207 526 IR 009 613

AUTHOR	Harris, Diana, Ed.; Nelson-Heern, Laurie, Ed.
TITLE	Proceedings of the NECC 1981. National Educational Computing Conference (3rd, North Texas State University, Denton, Texas, June 17-19, 1981).
INSTITUTION	Iowa Univ., Iowa City. Weeg Computing Center.
REPORT NO	ISBN-0-937114-014
PUB DATE	Jun 81
NOTE	361p.; For related document, see ED 194 060.
AVAILABLE FROM	Computer Science Department, University of Iowa, Iowa City, IA 52242 ($10.00).
EDRS PRICE	MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS	*Computer Assisted Instruction; Computer Assisted Testing; Computer Graphics; *Computer Managed Instruction; *Computers; *Computer Science Education; *Microcomputers; Programing Languages; Simulation; Videodisc Recordings
IDENTIFIERS	*Computer Literacy

ABSTRACT

This volume includes the texts of more than 50 papers presented at a conference which was organized to present in one forum all major work regarding computers in education in the United States, as well as abstracts of the special sessions, tutorials, and project presentations which took place at the conference. Among the topics covered in these materials are the following: simulations, videodisc project funding, administration, computer literacy, business, higher education, computer science, humanities, science, social science, preschool/elementary applications, graphics, mathematics, engineering, and health education. (Author/LLS)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED207526

U.S. DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

Proceedings of NECC 1981
National Educational Computing Conference

Edited by
Diana Harris
Laurie Nelson-Heern

Hosted by
North Texas State University
Denton, Texas

17, 18, 19 June 1981

"PERMISSION TO REPRODUCE THIS MATERIAL IN MICROFICHE ONLY HAS BEEN GRANTED BY

NECC
Diana Harris

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

IR009613

NECC 1981 Steering Committee

Alfred Bork
    University of California-Irvine
Richard Close
    U.S. Coast Guard Academy
Karen Duncan
Gerald L. Engel
    Christopher Newport College
Norman Gibbs
    College of William and Mary
John W. Hamblen
    University of Missouri-Rolla
Diana Harris
    University of Iowa
Harry Hedges
    Michigan State University
Lawrence A. Jehn
    University of Dayton
Sister Mary Kenneth Keller
    Clarke College
Doris K. Lidtke
    Towson State University
Elizabeth Little
    Swarthmore College
Sister Patricia Marshall
    Xavier University of Louisiana
Mike Mulder
    ESI, Inc.
Richard E. Pogue
    Medical College of Georgia
James Poirot
    North Texas State University
Nancy Roberts
    Lesley College
Theodore J. Sjoerdsma
    University of Iowa
Dennis Spuck
    University of Houston
E. M. Staman
    University of Missouri-Columbia
David L. Stonehill
    University of Rochester
David B. Thomas
    University of Iowa

General Chairman: James Poirot
Program Co-chairmen: Alfred Bork, Gerald Engel,
    Doris K. Lidtke, Nancy Roberts
Local Arrangements Committee: Tom Irby,
    Kathleen Swigger, Darrell Ward
Proceedings: Diana Harris

The National Educational Computing Conference wishes to thank the following people for their contribution of effort, time, and knowledge as referees for the papers submitted for presentation.

Emilie E. Attala, California Polytechnic State University
Richard Austing, University of Maryland
Bruce H. Barnes, National Science Foundation
Della T. Bonnette, University of Southwestern Louisiana
A. Martin Buoncristiani, Christopher Newport College
M. Kent Cueman, Weyerhauser Corp.
Elizabeth-Anne Daly, Christopher Newport College
William Dorn, University of Denver
Karen A. Duncan
B. G. Dunn, Fairmont State College
James Eagle, Christopher Newport College
Gerald Engel, Christopher Newport College
Selby Evans, Texas Christian University
F. T. Fink, Michigan State University
David Game, Christopher Newport College
Norman Gibbs, College of William and Mary
Sheldon P. Gordon, Suffolk County Community College
Peg Guyder, Skidmore College
John Hamblen, University of Missouri, Rolla
Harry Hedges, Michigan State University
Hugh Hilliard, Christopher Newport College
A. A. J. Hoffman
Larry Jehn, University of Dayton
Sherryl Johnson, Colgate University
Vincent H. Jones, Southern University
Gary Kessler, St. Michael's College
Dan Klassan, St. Olaf College
Joyce C. Little, Community College of Baltimore
Doris Lidtke, Towson State University
Richard W. Lott, Bentley College
David E. Maharry, Wabash College
Walter Maner, Old Dominion University
Robert Mathis, Old Dominion University
Donald H. McClain, University of Iowa
John McGregor, Christopher Newport College
Edmund D. Meyers, Jr., Boys Town
Andrew Molnar, National Science Foundation
Michael Moshell, University of Tennessee
Michael Mulder, ESI, Inc.
Robert Nooman, College of William and Mary
Linda Petty, Hampton Institute
Richard Pogue, Medical College of Georgia
James L. Poirot, North Texas State University
William Porterfield, Hampden-Sydney College
Alice F. Randall, Christopher Newport College
Leroy Roquemore, Southern University
Hassell A. Simpson, Hampden-Sydney College
Ted Sjoerdsma, University of Iowa
David Thomas, University of Iowa
Robert A. Thomson, University of Dayton
Rita Wagstaff, Cuyahoga Community College
Roger Weiss, University of Dayton
Gary Wittlich, Indiana University
Frank Wojick, Virginia Institute of Marine Science

And special thanks to the authors whose cooperation kept this publication on schedule. --Diana Harris

Foreword

The 1981 National Educational Computing Conference (NECC-81) builds on the success of the two previous conferences held in Iowa and Virginia. NECC provides a broad forum for discussion among individuals at all levels with interests in educational computing. As a cooperative venture undertaken by 15 professional organizations, NECC-81 has as one of its main objectives presenting in one forum all major work regarding computers in education in the United States. This volume represents the papers presented at the conference and abstracts of the Special Sessions and Project Presentations.

The reviewed papers of this conference represent the diverse interests of researchers throughout the country. Gerald Engel of Christopher Newport College, program chairman, along with the many reviewers across the country deserve our thanks for the high quality of this part of the conference.

Doris Lidtke of Towson State University coordinated the special sessions and tutorials of the conference. She and the individuals participating in these activities receive our thanks.

Nancy Roberts of Lesley College organized a new feature of the conference this year, the Project Presentations. These presentations, representing ongoing research and development of educational materials, allow for sharing of innovative ideas and materials. Alfred Bork of the University of California at Irvine coordinated the pre-conference workshops, also a new feature of NECC.

Very special thanks go to Diana Harris of the University of Iowa who edited these proceedings. Thanks are due also to the entire NECC-81 Steering Committee for their excellent guidance in preparing for the conference. Sincere appreciation is also expressed to all those hard working individuals at North Texas State University for hosting NECC-81.

Finally, we thank all those individuals who came to NECC-81 and helped to ensure the success of this conference.

James L. Poirot
Chairman, NECC-81
North Texas State University
Denton, TX 76203

Table of Contents

SIMULATION

1 A Computer Simulation Model for Teaching Intermediate Macroeconomics
John E. Silvia
4 BIZNES: A Successful Approach to Simulation in Economic Education
Robert Schenk
8 Teaching Experimental Design with a Mathematical Model Algorithm
John A. Ward

SERVICES

12 Protecting Interactive Programs from User Input Errors
Robert T. Maleske
18 Transportability of Instructional Computer Programs: Issues and Examples
Eugene A. Herman
24 The Next Step: The Use of the MicroSynergistic (Microcomputer Cluster) in Education
John W. Motto

SPECIAL SESSIONS

34 Funding of Videodisc Projects
Chaired by Alan Breitler
35 Computing Concepts in Elementary Schools
Chaired by Margaret Christensen

TUTORIAL

36 Program Development Methodology
A. J. Turner

PROJECT PRESENTATION

37 College CAI Projects in the Basic Skill Areas
Susan M. Wood, George M. Bass, Jr., Roger R. Ries, Helen Heller, S. A. Eveland, and Michael G. Southwell

ADMINISTRATION

39 Computer-Assisted Academic Departmental Scheduling
Ronald Prather and Ahcene Rabia
45 A Financial Planning Model
Robert L. Horton and Gary L. Eerkes
49 A User-Oriented Classroom Use Program for Colleges and Universities
Larry Luce

COMPUTER LITERACY INTEGRATED IN THE CURRICULUM

59 Developing Computer Literacy at a Liberal Arts College
Nancy H. Kolodny and Gene Ott
66 A Novel Approach to Computer Literacy for Natural Science Students
R. V. Rodriguez and F. D. Anger
74 Integrating Computing Literacy into Existing Curriculum: Some Case Studies
Rachelle S. Heller and C. Dianne Martin

SPECIAL SESSIONS

78 Research and Application of Videodiscs
Chaired by Richard A. Pollak
80 Computer Literacy for Elementary School Teachers
Chaired by Jacques LaFrance

PROJECT PRESENTATION

81 Elementary School Applications: LOGO
Pat Lola, Coleta Lou Lewis, Theresa Overall, Henry Gorman, Jr., Kay Murphy, and Daniel H. Watt

BUSINESS

83 Students Confront Data Base
Doris Duncan Gottschalk and Roy Elliott
86 Physiology of Small Data Processing Systems
Andrew Vazsonyi
89 Computer-Simulated Management Training Better than Case Studies
Chauncey Burke and Robert E. Callahan

EDUCATION

94 The Role of a Computer-Based University-Wide Testing Service to Manage Large Enrollment Courses
Lewis J. Wood and Jacqueline Ortega
99 A Study of the Effectiveness of Computer-Assisted Feedback for Modifying Teacher Behavior in an Inservice Setting
Ted S. Hasselbring and Cathy L. Crossland
105 Computing as a Way of Brainstorming in English Composition
Hugh Burns

SPECIAL SESSIONS

109 Graphic Design Issues in Computer-Based Education
Raymond Nichols, Jessica Weissman, and Brian Shankman
111 Evaluating and Selecting Computer Studies Textbooks: Art, Science, or Chance?
Chaired by Stephen Mitchell

TUTORIAL

112 Structured Systems Analysis Tutorial
V. Arthur Owles and Michael J. Powers

PROJECT PRESENTATION

113 College CAI Projects in Content Areas
Steven V. Bertsche, James D. Spain, James W. Parrish, Barron Arenson, and Matthew Wozniak

COMPUTER SCIENCE I

115 Simulation of Computer Architecture Using a Hierarchical Computer Hardware Description Language (CHDL)
W. A. Skelton and R. S. Walker
121 Opal--A Hardware-Independent Assembly Language for Education
Thomas G. Luce
127 Pre-Assembler: A Bridge (Course) Over Troubled Waters
Ronald C. Turner and Roland J. Keefer

HUMANITIES

132 Computer Assistance in the German Individualized Instruction Program at the Ohio State University
Heimy F. Taylor
136 Computer-Assisted Instruction in Melodic Composition
Paul E. Dworak
142 Computer-Assisted Instruction in Music: Analysis of Student Performance 1973-1980
Wolfgang Kuhn and Paul Lorton, Jr.

SPECIAL SESSIONS

149 Computers and Continuing Health Education
Richard Pogue, Ronald Comer, James E. Eisele, Lynn L. Peterson, and Thomas Held
150 Graphics and Computer-Based Learning
Chaired by Eugene Herman

PROJECT PRESENTATION

152 Continuing Education
Frank A. Settle, Michael Pleva, Sally J. Weiler, George Mozes, and Lebert R. Alley

COMPUTER SCIENCE II

154 A Successful Experience with Programming Languages at a Liberal Arts College
Linore Cleveland
161 The C-Basic Interpreter: A Learning Tool for Compiler Construction
John A. Rohr
169 Teaching Fundamental Computer Technology by Subject Matter Professionals Using Programmable Calculators
Reed D. Taylor and Michael W. Woolverton

SCIENCE I

175 An Educational Computer Network for the Users of Chemical Instrumentation
Michael A. Pleva and Frank A. Settle, Jr.
179 Newton -- A Mechanical World
Alfred Bork, Stephen Franklin, Martin Katz, and John McNelly
184 Design of Computer-Based Instructional Materials to Enhance the Problem-Solving Abilities of Science Students
Jerry P. Suits and J. J. Lagowski

SPECIAL SESSIONS

190 MicroSIFT Project and Courseware Evaluation
Robbie Plummer
191 The Possible Effect of Color on Computer-Assisted Learning
Chaired by Herb Nickles

TUTORIAL

192 Pascal Tutorial
H. P. Haiduk

PROJECT PRESENTATION

193 Precollege CAI
Diana B. Radspinner, Diann Dalton, Arlene M. Smith, Janet M. Scott, Deborah C. Slaton, Katherine L. Helwick, Vicki S. Smith, Connie J. Brooks, and M. Beatriz Beltran

COMPUTER SCIENCE III

195 Software Documentation for Student Projects
John A. Rohr
198 Firmware Development
Robert N. Cook
206 Congition and Programming: Why Your Students Write Those Crazy Programs
Elliot Soloway, Jeff Bonar, Beverly Woolf, Paul Barth, Eric Rubin, and Kate Ehrlich

SCIENCE II

220 Science Literacy in the Public Library -- Batteries and Bulbs
Arnold Arons, Alfred Bork, Stephen Franklin, Barry L. Kurtz, and Francis Collea
225 Teaching Simulation Techniques with Microcomputers
J. D. Spain
228 Computer-Assisted Instruction in Secondary Physical Science: An Evaluation of Student Achievement and Attitudes
Gary H. Marks and Rolland Bartholomew

SPECIAL SESSIONS

238 An Overview of Micro-CMI
Don McIsaac
239 Network Information Resources for Computers in Teaching and Learning
Chaired by Karl Zinn

PROJECT PRESENTATION

240 Precollege Computer Literacy
Ronald E. Anderson and Edward E. Wachtel

SPECIAL SESSIONS

242 Report of Activities of the ACM Elementary and Secondary Schools Subcommittee
Chaired by J. Philip East and Jean B. Rogers
243 The Future of Microcomputers in Education: A Vendor's View
Glenn Polin
244 Personal Computing to Aid the Handicapped: The Johns Hopkins First National Search
Paul L. Hazan

SOCIAL SCIENCE

245 The Study of U. S. Regional Shifts, Metropolitan Growth and Neighborhood Structure with Census Microdata
Harold Benenson and Steven Just
254 A Computerized Course in Elementary Statistics: Educational Objectives and Methods
Richard W. Evans
259 Games People Will Play: Development of a Method to Assess Interest in Instructional Games
Jerry D. Neideffer and Selby H. Evans

PRESCHOOL/ELEMENTARY

264 Computers and the Nursery School
Kathleen M. Swigger and James Cambell
269 Computer-Assisted Instruction in Reading Used for Kindergartners and First Graders in Dillon, Montana, 1979-1980
Nellie Bandelier
272 A Study of Preschool Children's Use of Computer Programs
Coleta Lou Lewis

SPECIAL SESSIONS

276 Projects Supported through National Networking
Chaired by Paul S. Heller
277 Guidelines for the Successful Selection and Operation of Minicomputers and Microcomputers
Douglas S. Gale
278 Computer-Based Education: Strategies for Success
Franz E. Fauley

PROJECT PRESENTATION

279 Precollege Software Evaluation and Administration
Karen Billings, Daniel H. Watt, Ted Mims, and Charles G. Boody

GRAPHICS

281 Some Uses of Computer Graphics in the Calculus Classroom
Robert F. Maurer and Timothy P. Donovan
284 The Human Factors of Color Display-Based Instruction
John Durrett, Catherine Zwiener, and Robert Freund
287 Graphics in Computer Science
Freeman L. Moore

TEACHER DEVELOPMENT FOR COMPUTER LITERACY

292 Microcomputers in Education: A Course in Computer Literacy for Educators
Cheryl A. Anderson
297 Microcomputer Classroom Use Patterns
Dorothy H. Judd
302 Educating Urban Elementary Teachers in Computer Science
Tim Carroll and Nancy Johnson

SPECIAL SESSION

309 Microcomputer Networks
Robert F. Tinker and Richard E. Pogue

TUTORIAL

310 Ada Tutorial
Robert F. Mathis

PROJECT PRESENTATION

311 College Computer Literacy
Edward B. Wright, T. P. Kehler, M. Barnes, James W. Garson, and David Miles

MATHEMATICS

313 Applications of the Definite Integral and Basic
Peter A. Lindstrom
320 Starting a Computer-Based Learning Project
Barry MacKichan, J. Mack Adams, and Roger Hunter
324 A Computer-Based Dialogue for Developing Mathematical Reasoning of Young Adolescents
David Trowbridge and Alfred Bork

ENGINEERING

327 Switching Over to the Micros
W. D. Maurer
332 Facilitating Small-System Modular Instruction with an Information System
Amanda Evans and David M. Himmelblau
336 Personal Computer Simulation Programs as Teaching Aids: A Successful Experiment in Undergraduate Electrical Engineering Education
Aart J. de Geus, Leo R. Pucacco, and Ronald A. Rohrer

PROJECT PRESENTATIONS

341 College Mathematics
Robert F. Maurer, Timothy Donovan, Wayne F. Mackey, Susan Lindsay, Harold Harp, Thomas Wallgren, and LaRuth H. Morrow

343 Author Index

345 Subject Index

A COMPUTER SIMULATION MODEL
FOR
TEACHING INTERMEDIATE MACROECONOMICS

John E. Silvia
Indiana University - Indianapolis

INTRODUCTION

A computer simulation model is an aid used to teach intermediate macroeconomics. Three characteristics distinguish this model from others. First, it is programmed using real world data so its parameters and exogenous variables represent actual economic relationships. Second, the model incorporates theory at an intermediate level. It is not a simplified algebraic "Principles of Economics" model nor does it use extremely complex real world equations. Several important theoretical developments (e.g., the permanent income hypothesis and aggregate supply) are incorporated, too. Third, the model offers the student a number of decision-making opportunities. Many models are used by the student with an economic goal, policy option, and specific assignment given by the instructor. By using this, the student can choose from a set of economic goals and a number of policy tools.

We use the model to develop student awareness of how the economy operates (theory) and how decision-makers attempt to make the economy achieve desired objectives (policy). The model simulates the U.S. economy. It is programmed for use on a DEC-10 computer and used interactively by the student. The classroom experimentation with the model began in the spring 1980 semester. The simulation approach uses three techniques: case study, role-playing, and model building. A series of lessons are given; the student plays the role of economic policy decision-maker and uses the model to achieve certain economic goals. By using all three techniques, this excerise goes beyond a machine-tutorial.

With a properly developed instruction model, several desirable results are achieved. First, students become more involved in the learning process. Second, students develop positive attitudes toward the subject matter and education in general. Third, skills of critical thinking and knowledge of economic analysis are increased.

THE MODEL AND LESSONS

The simulation model is a set of equations which give a solution for all endogenous (determined within the economic model) variables within this set of equations. Given a theoretical model, decisions must be made about what parameters are to be used, which variables are taken as exogenous, that is determined outside the model, and which variables are to be determined within the system.

Here are the key equations in the model and how the approach used is different from many others:

1. The potential output equation determines the long-term potential growth or output of the economy. This potential output depends, in part, on last period's investment and unemployment rate. Changes in current economic variables affect the long-term growth of the economy.
2. The aggregate demand sector is composed of several equations. The consumption equation uses the permanent income hypothesis. The investment equation uses a cost of capital variable and allows changes in corporate tax policy and depreciation allowances to affect investment. Government spending is exogenous, but tax revenues are endogenous. The monetary base is exogenous, but the money supply is endogenous.
3. The endogenous variables include, for example, consumption, investment, gross national product, and several interest rates. The unemployment rate and inflation rate are also

determined within the model.

4. The exogenous policy variables include government spending, tax rates on personal and corporate income, the monetary base, the discount rate, and the reserve requirements.
5. The international sector incorporates the effects of GNP, inflation, and exchange rates on imports and exports.

A unique aspect of the model is the built-in flexibility. Certain equations can be replaced by others, reflecting a slightly different theoretical approach. This is true of the inflation equations and aggregate supply equations. These alternative specifications allow a comparison of how a given policy may have different effects on the endogenous variables under different theoretical regimes.

HOW WILL STUDENTS USE THE MODEL?

The average student registering for the intermediate macroeconomics course is a junior with one computer science course that introduces computer data processing and some Fortran programming. The average student has also had two semesters of principles of economics, but no statistics or econometrics.

With this background, student interaction with the model should involve little or no Fortran programming. The model is written in Fortran IV on the DEC-System 10 at IUPUI. Students use the model through an interactive typewriter terminal, DEC-10, using very simple log-on/log-off procedures.

Students are introduced to the model in a group session. They proceed through a series of computer questions and student responses, and at the end of this session, each student receives a hard-copy of the complete model and all information needed for future exercises.

The model is introduced halfway into the semester so that students have completed a significant amount of the economic theory in the model. When the model is introduced is an important decision: it should be done after enough theory has been developed so that students understand the equations and feel comfortable with the model. However, the model cannot be introduced so late that it appears as a late-comer to the course or does not allow students enough time to significantly improve their understanding and attitudes.

The student assumes the role of policymaker and has raw data for the economy up to the starting point of his reign as decision-maker. Also, the student has the model's equations and a list of policy tools. The model and policy tools are defined to closely match the intermediate theory familiar to the student.

In the simulation, a series of six lessons focuses on the theory of how the economy operates. The first lesson requires the student to trace through the equations of the model the impact of increased government spending. Each student hands in a written essay explaining the policy change and its impact on selected economic measures (e.g., inflation, interest rates, or unemployment). A second part of the first lesson examines alternative specifications for the inflation equation. A third section examines Okun's Law. This exercise reemphasizes economic theory as part of the model and reinforces the textbook theory.

For the following lessons, the student must look at the present economy, define what he sees as the major problem or problems, suggest a policy, and then run the simulation to analyze the policy effects. Each lesson requires writing an essay graded by the instructor.

The second lesson uses a simplified Keynesian model, the third, a monetarist model, and the fourth, an empirically-based post-war model. The Keynesian and monetarist models are stylized, but have a flexibility that allows for demonstration of certain likely policy outcomes for the respective programs. For each model the student retains the right to choose his own policy goal and policy tools.

The fifth lesson examines the importance of the international sector. Students attempt to offset exogenous shocks or examine the impact of those shocks on the domestic economy.

The sixth lesson examines the distinction between short-run and long-run price and output effects.

Within each lesson, there are a number of prompts and options that check the student's policy design. Errors in data entry or incorrectly typed information do not abort the computer run. It is important that students not face an aborted run. The computer program attempts to help the student complete the lesson rather than punish for incorrect typing.

EVALUATION OF STUDENT OUTCOME

In the experiment currently used and under evaluation, students have been paired by selected characteristics into two groups. Only the experimental group uses the computer model, but both the control group and the experimental group attend the same lectures and are tested with the same exams.

At the beginning of the semester a pre-test is administered to all students in the class. The pre-test covers both cognitive and attitudinal aspects with regard to macroeconomics. A post-test is administered at the end of the semester.

The pre-test has three parts. The first part asks for background information: age, sex, major, overall grade point, grade point in economics, economic courses taken, and student attitudes towards computers. Based upon this information, students are chosen for either control or experimental group. Members of the groups are paired as closely as possible.

The second and third parts of the pre-test are surveys of attitude and economic knowledge respectively. The student is queried about personal opinions of school, computers, and economic issues. The last part determines the student's economic knowledge.

The post-test is composed of the third and fourth parts of the pre-test. The assumption is that the students' attitudes toward and knowledge of economics and computers has improved. Therefore, the experimental groups' attitudes and economic knowledge should be statistically improved over the control group.

The effectiveness of the model is evaluated using a linear regression model. The model is:

$$Y = \alpha_0 + \sum_{i=1}^{n} \beta_i X_i + \gamma Z + \varepsilon$$

where Y = student performance
X_i = independent variables
Z dummy variable = 1 experimental group
= 0 control group

The regression results appear in Table I. The dependent variable, GAP, is the proportion of pre- to post-aptitude test gap closed = (Post-Pre/40-Pre). The second dependent variable, attitude change (ATCH), is measured by the proportion of pre- to post-Penn State questionnaire gap closed (Post-Pre/200-Pre). The independent variables are student grade point average (GPA), age (AGE), major (M), pre-course attitude toward economics (ATT), student effort (EFF). The pre-course attitude toward economics was measured using the Penn State University Questionnaire. Student effort was measured by the number of hours spent on the course over the semester.

Table I

Educational Performance of Students
(t statistics in parentheses)

$$GAP = \underset{(1.03)}{0.30} + \underset{(2.99)}{3.83}\,GPA + \underset{(1.12)}{0.80}\,AGE + \underset{(1.04)}{0.76}\,M + \underset{(2.43)}{1.2}\,ATCH + \underset{(1.88)}{2.19}\,EFF + \underset{(2.10)}{2.72}\,Z$$

$$ATCH = \underset{(2.54)}{0.33} + \underset{(2.57)}{0.28}\,GPA + \underset{(1.48)}{0.64}\,AGE - \underset{(2.32)}{0.56}\,M + \underset{(2.81)}{4.42}\,GAP + \underset{(1.44)}{1.23}\,EFF + \underset{(1.93)}{1.09}\,Z$$

The regression results show that the independent variables explain a significant proportion of student cognitive and attitudinal performance. For the GAP measure the impact of the simulation model (Z) is positive and statistically significant. Students do find the model useful in learning the theoretical underpinnings of the macroeconomy. Students also learn by comparing alternative model specifications and using the model to forecast the economy's performance for different policy options. Student cognitive performance is also positively affected by GPA, AGE, ATT, and EFF. These results support those seen in other studies.

Student attitudes are positively affected by the simulation model and the effect is statistically significant. Attitude change appears related to GPA, AGE, and MAJOR.

SUMMARY

This paper has briefly presented some of the evidence on the design and experimental results of a simulation model used in teaching intermediate macroeconomics. The model is developed using intermediate theory, is placed on the computer, and is used by students to complete a series of lessons. The statistical results of the experiment show that using the model does have a positive impact on student cognitive and attitudinal performance.

Future research will focus on the improvement in predicting overall performance. The original study generated much data not yet tested, and it will be analyzed to seek better answers to the initial hypothesis.

BIZNES: A SUCCESSFUL APPROACH TO SIMULATION IN ECONOMIC EDUCATION

Robert Schenk
Saint Joseph's College
Rensselaer, IN 47978

This paper describes Biznes, an interactive simulation package designed for use in introductory microeconomics.[1] This package differs in approach and subject matter from simulations commonly used in economic education such as macroeconomic simulations and batch business games.

THE APPROACH

The approach taken in Biznes has three features not always used in economic simulations, but which contribute to the success of the package.

First, the simulation is not seen as a support for a traditional textbook but as a separate and equal approach. In the four weeks spent on the simulation in class, no other text except a 34-page student guide is used because the logical and natural ways to develop the course material using the simulation are different from those in textbooks.

Introductory courses traditionally approach microeconomics by exploring questions about what gets produced, what the resources and techniques are, and who gets the end products; or by exploring questions of economic efficiency: under what conditions does an economic system satisfy existing wants as well as they can be satisfied given technology and resources, and how close might the real world come to meeting these conditions? Though both of these approaches are useful and valid, they are not the only legitimate approaches as recent publications of nontraditional texts indicate. In traditional approaches the central focus is on the market, and behavior of individuals and firms is of interest only for what it reveals about markets. Further, the type of market stressed in traditional approaches is one in which no individual can have a noticeable effect.

The Biznes package deemphasizes the market and emphasizes the firm. Though a simulation of a market has a limited role, if any, for a policy-maker, a simulation of a firm can be totally dependent on the policy-maker. Simulations work best as an instructional tool when there is a role for policy. Though an approach emphasizing the firm leads away from some traditional concerns of microeconomics, it points to other issues which are usually difficult to fit into the course: discussion of the many simplifying assumptions economists make, X-efficiency, non-profit organizations, and the production function, to name a few.

Second, the model used in the simulation is kept very simple. It is based on the three boundaries or limitations that a firm faces in its search for profit: the supply curves of resources, the production function, and the demand curve. Any theory of the firm must be based on these three boundaries, and simplicity is achieved by assuming that the boundaries are fixed, a conventional assumption in economics. The student guide has several pages listing reasons why the boundaries might not be fixed in the real world. The view that the firm is limited by three sets of boundaries is usually obscured in introductory courses that focus on markets. Textbook authors often do not explain that in output markets a total cost curve comes from combining the production function with the supply curves for resources or that in the resource markets a revenue product curve depends on the demand curve and the production function.

In discussing the negative findings about the effectiveness of computer simulations in economics, Siegfried and Fels suggest that the temptation to do complex things may lead authors away from instructional objectives.[2] Biznes does not suffer from this problem.

Third, the simulation does not stand

EXHIBIT ONE: MAJOR INSTRUCTIONAL OBJECTIVES

After completing this package, the student should be able to:

1. define elasticity of demand, marginal product, production function, marginal revenue, isoquant; (65-90)
2. explain what the law of diminishing returns is and under what conditions it holds; (50-60)
3. compute marginal product and approximate marginal cost when given levels of an input, output, and total cost; (30-55)
4. calculate marginal revenue if given price and elasticity of demand at that price; (50-75)
5. use the equimarginal principle to reach positions of minimum cost when given marginal products and prices of two inputs; (40-75)
6. explain why the demand curves and total cost curves can be treated as boundaries; (40-60)
7. list at least five assumptions that are made in the economic theory of the firm that distinguish it from real-world firms; (40-70)
8. compute marginal revenue product from marginal product and marginal revenue; (50)
9. decide what changes in levels of inputs will lead to higher profits when given data on marginal resource cost and marginal revenue product. (40)

---

*Note: numbers in parentheses represent the approximate percentage of students accomplishing each objective, with estimates based on test results.

alone but is complemented with a student guide that explains the objectives and provides written exercises.[3] A list of the objectives is given in Exhibit 1.

The written exercises are a vital part of this package. They provide guidance that students need to see what they are to learn from the simulation. Most of the goals listed in Exhibit 1 can be achieved by working through the written assignments. A sample output of the simulation itself, given in Exhibit 2, shows that the simulation itself does not guide students as to what is important. Without the exercises, the simulation would be of little educational use.

Another reason for written assignments is that students have a tendency to guess when they confront a difficult problem on a computer terminal. (For this reason, I believe that what can be accomplished with traditional CAI -- programmed learning on an interactive computer -- is quite limited.) Written assignments provide an incentive not to guess but to think. And thinking, applying ideas, is what economic simulations are supposed to encourage. A quotation from Don Patinkin summarizes what I am trying to say: "From this ... I learned the hard way that just reading and listening were not enough; that full understanding of the principles of economic analysis could be achieved only after sweating through their application to specific problems, with pencil and paper in hand."

A final advantage of giving written assignments is that it is a way of stressing to students that the material is important and worth their time.

RESULTS

This simulation package has been under development for a number of years. Only during the first semester of the 1980-81 school year did it reach a stage where it seemed to be successful. In this semester it was used as the basis of a four week unit of an introductory course. Students had to complete seven written assignments. I allowed students to work in groups of two, but only about 1/3 did so. I kept records of student use; each lesson required about 20 minutes per student of computer time. Though some time was also required to analyze computer results and to answer questions about them, this package seems no more demanding of student time than usual course assignments.

The conclusion of success is based largely on student evaluation and on subjective evaluation of the instructor. Though it is possible to measure how well each objective was met, it is very difficult to do a comparison to other methods of presenting the subject matter because,

EXHIBIT TWO: CORE OF COMPUTER OUTPUT

```
After reviewing these results, how many workers do you wish to hire for
the next period?  (You may hire fractional workers.)
4.3
How Many units of capital do you wish to use?  (Again, you may use
fractional units.)
5.6
You will produce 93  units of output.
What price would you like to charge? (Enter a number without a $.)
900

-------------------------------------------------------------------
      Labor hired:   4.3             Cost per unit:   8000
    Capital hired:   5.6             Cost per unit:   12000
 Amount produced:    93           Amount sold:   93

    Total Revenue:   83700         Per unit revenue:   900
    Total cost   :   101600          Per unit Cost   :   1092.47
    Profit       :  -17900        Per unit profit :  -192.473

The net value of your firm is now $ 962100.
-------------------------------------------------------------------

Will you pay $1000 for additional information?
y

          ----------------    Economist's Report   ---------------
          Marginal product of labor =  14.1443
          Marginal product of capital =  6.01522
          Elasticity of demand (at P= 900  ) =  4.17593
          ----------------------------------------------------------

Would you like to see what other information is available?
no

After reviewing these results, how many workers do you wish to hire for
the next period? (You may hire fractional workers.)
```

as far as I know, there are no other materials with quite the same goals. Estimates of the percentage of students that accomplish each objective, based on results from a test after the class completed the package, are given in Exhibit 1. The percentages have a wide range because questions testing the same objective differed in difficulty. The classes which used this simulation were predominantly first-semester freshmen with mean SAT scores near the national average.

Though there are those who refuse to consider any evidence other than that from experimental-control data, such evidence cannot be obtained for many questions involving instructional usefulness. For example, one cannot evaluate whether the various nontraditional texts are better than traditional texts because the goals involved are not the same. The experimental-control data is meaningful only when the goals are held constant.

An alternative way of measuring success is from student evaluations, though again it is more suggestive than conclusive. Exhibit 3 shows the two questions on a 40-question evaluation concerning the package. These results were more favorable than those given the text, the lectures, the class discussion, the course overall, and the instructor.

SUMMARY

To briefly summarize, students spent a month working with the simulation with the aid of a student guide. Features of this simulation package that were important to its success were its independence from a standard text, its simplicity, and the role it gave to the student guide. Finally, students said in the course evaluation that they believed the simulation and the student guide were very helpful parts of the course.

EXHIBIT THREE: STUDENT EVALUATION OF SIMULATION AND STUDENT GUIDE

	Strongly Agree	Agree	Uncertain	Disagree	Strongly Disagree
The computer simulation significantly contributes to this course.	34	33	6	4	2
The student guide for the computer simulation is a valuable part of this course.	32	29	12	3	3

FOOTNOTES

1.) The package described in this paper was developed with the partial support of a NSF LOCI Grant No. SER78-00065. It consists of a Basic program of 515 lines of code, a student guide, and instructor's notes. Further information about the package is available from the author.

2.) John J. Siegfried and Rendig Fels, "Teaching College Economics: A Survey," Journal of Economic Literature, Sept. 1979. p. 942.

3.) Two years ago I described two macroeconomic simulations that I was using. Since then I have developed a more complete student guide which has drastically improved the effectiveness of these simulations. See "Simulating The Great Depression in Introductory Macroeconomics," Proceedings of the National Educational Computing Conference, June 1979.

TEACHING EXPERIMENTAL DESIGN WITH A MATHEMATICAL MODEL ALGORITHM

John A. Ward
Incarnate Word College
San Antonio, Texas

INTRODUCTION

Computer simulations effectively permit students to perform experiments on systems that otherwise could not be studied in the laboratory due to time, cost, or size constraints.(1) They are available to educators through various sources, (2,3) and can be used to introduce students to the principles of scientific method and experimental design.(4)

In general, a computer simulation is based on a mathematical model, an equation that describes the behavior of a physical, biological, or social system. The equation is made up of variables, the values of which can be determined by the student or the computer program. Students investigate the system by changing the values of variables under their control and observing the effect.

This paper describes a general algorithm that can be used to teach students how to use mathematical modeling to design experiments.

SETTING

The program is used in conjunction with the Minority Biomedical Support (MBS) Program at Incarnate Word College. The MBS Program, an NIH-supported project, enables minority undergraduates to participate in research projects under the supervision of a faculty member. Since most of the students were working on cardiovascular research, Poiseuille's law was the model chosen.

Although Poiseuille's law strictly applies to the laminar flow of Newtonian liquids through rigid tubes, it is a reasonable approximation of the flow of blood through vessels.(5) According to Poiseuille's law,

Q=PI x (P1-P2) x R∧4/(k x N x L)

where Q is rate of flow,
P1-P2 is the pressure gradient,
R∧4 is the fourth power of the radius of the vessel,
k is a constant of proportionality,
N is viscosity, and
L is the length of the vessel.

The program is designed so that any algebraic equation can be used as a model, depending on the needs and interests of the students.

The model is presented to the students in seminar, using a demonstration-discussion approach.

INSTRUCTIONS

When the instructor starts the program, the following instructions are displayed:

```
*******************************************
                HEMODYNAMICS
*******************************************

IN THIS SIMULATION YOU WILL BE ABLE TO
DETERMINE HOW BLOOD PRESSURE, LENGTH,
RADIUS, AND VISCOSITY AFFECT THE FLOW OF
BLOOD THROUGH A VESSEL.

TO DO SO YOU WILL HAVE TO HOLD ALL BUT
ONE INDEPENDENT VARIABLE CONSTANT AND
PERFORM AN EXPERIMENT TO SEE HOW THAT
ONE INDEPENDENT VARIABLE AFFECTS THE
DEPENDENT VARIABLE, FLOW.

YOU WILL THEN GRAPH THE DATA, TRANSFORM-
ING VARIABLES IF NECESSARY, AND SEE IF
THE DATA FIT  A LEAST SQUARES LINE DRAWN
THROUGH THE POINTS.

...PRESS RETURN TO CONTINUE.
```

The instructor discusses the instructions with the students, clarifying where necessary. Extensive explanation is not needed at this stage since the process becomes explicit in the running of the program.

Next, a list of empirical values for the system variables (6) is displayed:

```
HERE ARE SOME VALUES FOR BLOOD VESSELS IN
THE MESENTERIC CIRCULATION OF THE DOG.

     MESENTERIC ARTERY
        RADIUS              0.15 CM
        LENGTH              6.0  CM
        PRESSURE DROP       0.8  MMHG

     CAPILLARIES OF VILLI
        RADIUS              0.00040 CM
        LENGTH              0.04    CM
        PRESSURE DROP       2.4     MMHG

     MESENTERIC VEIN
        RADIUS              0.3  CM
        LENGTH              6.0  CM
        PRESSURE DROP       0.05 MMHG

     VISCOSITY         0.03 to 0.04 POISE

...COPY THIS TABLE, THEN PRESS RETURN TO
CONTINUE.
```

This is intended to provide the students with some realistic values to apply in investigating the system.

EXPERIMENTAL DESIGN

The students are then asked to choose one of the independent variables for experimentation. In order to study a system only one independent variable can be changed at a time. All others must be kept constant. For illustration, the effect of changing the vessel's radius will be studied. Values from the empirical table shown above are used:

```
WHICH INDEPENDENT VARIABLE DO YOU WANT TO
CHANGE?

     1) PRESSURE
     2) RADIUS
     3) VISCOSITY
     4) LENGTH

?2
SET CONSTANT VALUE FOR PRESSURE?  .8
WE WILL MAKE 6 MEASUREMENTS.
WHAT IS THE HIGHEST VALUE OF RADIUS?  .3
WHAT IS THE LOWEST VALUE OF RADIUS?  .0004
SET CONSTANT VALUE FOR VISCOSITY?  .035
SET CONSTANT VALUE FOR LENGTH?  6
```

Once the last value has been entered, the computer calculates the value of the dependent variable at six equal intervals within the chosen range of the independent variable and displays the results:

```
RADIUS              FLOW

4.00000004E-04      3.82976017E-14
 .06032             1.98050948E-05
 .12024             3.126997E-04
 .18016             1.5760322E-03
 .24008             4.96999011E-03
 .3                  .0121176

PRESS RETURN TO CONTINUE.
```

The students then choose the type of graph desired from a list of eight standard graphs, including linear, reciprocal, semilog, log-reciprocal, double log, and double reciprocal.

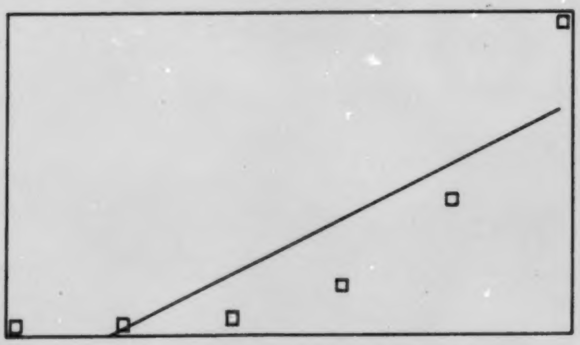
```
WHICH TYPE OF GRAPH DO YOU WANT?

     1) Y=MX+B
     2) Y=M/X+B
     3) LNY=MX+B
     4) LNY=M/X+B
     5) 1/Y=MX+B
     6) LNY=LNX+B
     7) Y=MLNX+B
     8) 1/Y=M/X+B

ENTER THE DESIRED NUMBER.  ?1
```

If the students do not have a sound theoretical basis for choosing a specific type of graph, a linear plot is suggested; the linear plot is the most familiar type of graph and most systems are piecewise-linear. That is, they can be approximated by a straight line over small ranges.

Next the values to be graphed are displayed. If a linear plot was chosen, these are the original values. When ready, the students direct the computer to graph the data. The computer then calculates the least squares line through the points, plots the line, and prints the equation of the line.



```
FLOW=.036573473 RADIUS+(-2.32731446E-03)
```

If a good fit is not obtained, the instructor suggests that the students plot another graph. At this point the students must understand the type of relationship expected between the dependent and independent variables. The instructor should help them explore the possible relationships before selecting the graph likely to give a good fit. It is important to let the students follow their intuition and discover the correct relationship by trial and error.

In this case, a logical conclusion would be that since an increase in radius will increase the cross-sectional area of the vessel, it will cause flow to increase. Flow would then depend on the second, or a higher power of the radius and a double-log plot would give a good fit.

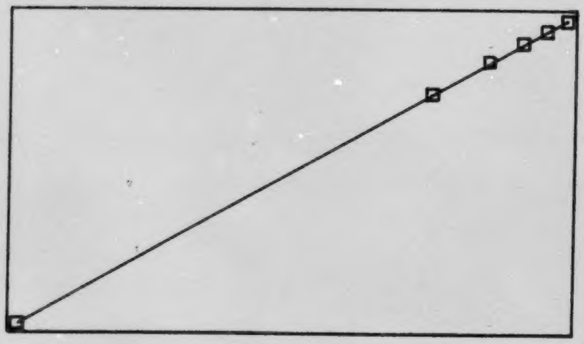
When a double-log plot is selected, the computer transforms variables and displays the results:

```
LN RADIUS       LN FLOW

-7.824046       -30.8933891
-2.80809156     -10.8295713
-2.11826553      -8.07026726
-1.71390993      -6.45284486
-1.42678308      -5.30433743
-1.2039728       -4.41309634

PRESS RETURN TO GRAPH DATA.
```

When ready, the computer displays the graph and the equation of the least squares line.



```
LN FLOW=4 LN RADIUS+(.402794844)
```

The instructor then explains what a good fit to the transformed variables means. In the case of the data fitting the equation:

Ln FLOW = 4 Ln RADIUS + CONSTANT,

the instructor takes the antilog of both sides to obtain the equation:

FLOW = CONSTANT X RADIUS ^ 4.

The flow is proportional to the fourth power of the radius. The coefficient of the radius in the logarithmic equation becomes the exponent of the radius in the linear equation. This leads naturally to a discussion of the profound effect that contraction or relaxation of the concentrically arranged smooth muscle in the wall of a vessel will have on blood flow. Doubling the radius of a vessel will increase flow by a factor of sixteen.

The instructor also explains that the constant of proportionality will depend on the dimensions used for the variables. If pressure were measured in dynes/sq.cm. instead of mmHg, a different value would have been obtained.

Finally, the instructor cautions the students against extrapolating the results beyond the range of values chosen for the independent variable, explaining that since the system was not tested outside that frame of reference, it cannot be assured it will behave the same way. This can be dramatically illustrated by a linear plot of the data obtained when a range of 0.14 to 0.15 cm is used for the radius.

CONCLUSIONS

The program described above effectively familiarizes students with mathematical models and their use in experimental design. It has been modified to analyze empirical data gathered in the MBS research project and used to test how well a hypothetical model fits the data. Mathematical modeling has provided a conceptual framework for experimental design.

In the classroom, the program has been used with other models to teach different concepts. For example, in Cell Physiology, the models shown in Table 1 are substituted in the program to teach cell growth and enzyme kinetics. The program has also been used as a prototype for the development of computer-assisted instruction in physical chemistry. Being able to change variables and instantly see the results graphed helps develop an intuitive understanding of system behavior.

Table 1. Mathematical models used in the general algorithm.

I. Hemodynamics

Y(I)=B(1)xPIxB(2)^4/(8xB(3)xB(4)

Y(I)= Flow

Variable	Graph
B(1)=Pressure	1) Y=MX+B
B(2)=Radius	6) LNY=LNX+B
B(3)=Viscosity	2) Y=M/X+B
B(4)=Length	2) Y=M/X+B

II. Bacterial Growth

Y(I)=EXP(B(1)xB(2)xLOG2+LOG(B(3)))

Y(I)= Cell Count

Variable	Graph
B(1)=Time	3) LNY=MX+B
B(2)=Growth Rate	3) LNY=MX+B
B(3)=Initial Count	1) Y=MX+B

III. Enzyme Kinetics

Y(I)=(B(5)xB(2))xB(1)/(((B(4)+B(5))/B(3))+B(1))

Y(I)= Velocity

Variable	Graph
B(1)= Substrate concentration	8) 1/Y=M/X+B
B(2)= Enzyme concentration	1) Y=MX+B
B(3)= K1, rate constant	8) 1/Y=M/X+B
B(4)= K2, rate constant	5) 1/Y=MX+B
B(5)= K3, rate constant	8) 1/Y=M/X+B

REFERENCES

1. Doerr, C. Microcomputers and the 3 r's. Rochelle Park, NJ: Hayden Book Company, Inc.; 1979:71-93.

2. CONDUIT. Catalog of CONDUIT reviewed and tested materials. Pipeline 5:21-36; 1980.

3. Ahl, D. H., editor. Basic computer games. Morristown, NJ: Creative Computing Press; 1979.

4. Bork, A. Computers as an aid to increasing physical intuition. Am. J. Phys. 46:796-800; 1978.

5. Mountcastle, V. B. Medical physiology. 13th ed. Saint Louis, Mo. C. V. Mosby Co.; 1974:914-917.

6. Schleier, J. Der Energieverbrauch in der Blutbahn. Arch. Gesamte Physiol. 173:172; 1918.

PROTECTING INTERACTIVE PROGRAMS FROM USER INPUT ERRORS

Robert T. Maleske, Ph.D.
Carthage College

Advances in hardware technology resulting in reduced size and cost have placed computers in the hands of instructors in all academic disciplines and at all levels of instruction. Interactive programs designed specifically for the instructional environment are being authored at an increasing rate, but the sophistication, appropriateness, reliability, and availability of this software continue to lag far behind these same attributes of the hardware. As a result, more and more instructors are becoming involved in all aspects of the software development process, including the initial conceptualization of an application, the logical design of the interactive sequence, and in some cases the actual coding of the program itself, i.e., the writing of the program in a given high-level language such as Basic, Fortran, Pascal, etc. Instructors are also taking on responsibility for selecting and evaluating software which is available either commercially, through a growing number of user groups, or from a variety of other sources. Since this trend is likely to continue, individuals having such responsibilities or interests will have a crucial role in controlling the quality of instructional software.

This paper tries to increase awareness regarding one of the common yet often overlooked problems inherent in many interactive instructional programs, and provides some concise and practical recommendations for teachers who are in a position to identify and reduce this problem. The results of a study on user attitudes and interactive response patterns involving 70 undergraduates at a four-year liberal arts institution are included as evidence of that problem.

BACKGROUND

The design and development of computer programs has traditionally been the almost exclusive property of individuals trained specifically for the purpose, i.e., computer programmers. However, the steadily increasing use of computers in the academic environment has brought with it an ever growing number of high school and college graduates who have combined one or two fundamental courses in programming--usually Basic or Fortran--together with a high level of intrinsic motivation and self-instruction. In short, programming skills are no longer exclusive to individuals formally trained in computer programming. The appearance of computers everywhere--including the home--has further contributed to the spread of self-taught programmers. Teachers at all levels and in all disciplines are well represented in this population.

This development places the instructional potential of computer technology more directly under the control of those who are most familiar with pedagogical technique and everyday practice, i.e., the teachers themselves. Formally trained programmers are highly skilled in programming languages and problem flowcharting, but often demonstrate limited sensitivity to the needs, attitudes, competencies, and behaviors of individuals who ultimately must interact with the programs which they, the programmers, have written. This limited sensitivity to the user has especially been a problem in instructional environments where students having limited knowledge of input format requirements and computer-specific command words are required to interact with the computer. Student interaction is often terminated due to misunderstandings regarding input requirements which could be avoided. Such problems with language have become common complaints, acquiring the status of buzz words: user orientation, human-machine interfacing, etc.

Most programmers, whether professional or self-taught, fail to go the extra mile when it comes to avoiding common and easily corrected programming flaws that disable the interaction between the user and the software. Teacher involvement in software acquisition and development processes is no guarantee that this general problem will be solved. However, given their focal position in the instructional environment, their awareness of the problem and strategies to avoid or correct it is important.

A significant body of literature regarding human factors in hardware and software design is developing. Much of this research is summarized in two publications: *Software Psychology: Human Factors in Computer and Information Systems*, Shneiderman (1980); and *The Psychology of Computer Programming*, Weinberg (1971). Much of this research centers around the study of programmer behavior. Some studies have assessed user preferences and user satisfaction, e.g., Gaines and Facey (1975); and several authors have provided general guidelines for designing interactive programs: Hansen (1971); Wasserman (1973); Pew and Rollins (1975); and Cheriton (1976). A review of this literature provides an excellent understanding of the focal issues involved in designing interactive programs. However, the specific problem of user-software interaction breakdown is given relatively little attention. Wasserman (1973) considers this problem in one of his five guidelines for designing "idiot-proof interactive programs." He specifically suggests that one should, "Provide a program action for every possible type of user input." Hansen (1971) also includes this problem among his three principles for user engineering; i.e., he states the principle of "engineering for errors." This paper targets the problem of user-software interaction breakdown for consideration by those involved in the instructional environment.

SPECIFICATION OF THE PROBLEM

The initial stages of interaction between the student and the program are crucial to keep the interaction going. Any response or no response from the student must be followed by unambiguous and easily understood computer output such as an informative statement or a request for input. When such statements are not easily understood, are ambiguous or misunderstood, serious problems which obfuscate the intended purpose of the program are likely to occur. Among these problems are:

1. The student may terminate the interaction immediately, not knowing what to do next.
2. The student may become frustrated after trial-and-error attempts to understand the output. The student may then terminate the interaction or--even if trial-and-error efforts eventually succeed--acquire a distaste for computer interaction.
3. The student may continue the interaction on the basis of a misunderstanding of the output or on the basis of false assumptions regarding the effect of his or her input. Such false assumptions are especially problematic when the program execution continues on the basis of input which the user did not intend. Examples:
    a. logical errors - student types "YES" instead of "NO."
    b. typographical errors - student types "YRS" instead of "YES."
    c. format errors - student types "(YES)" instead of "YES."

Problems of the first and second type can be reduced by wording and displaying all output as clearly and unambiguously as possible. You cannot pay too much attention to this aspect of interactive design. Furthermore, this is one area in which individuals with even minimal programming skills can greatly improve the quality of existing software by modifying the content and the appearance of the output display.

Another strategy for dealing with those problems is to provide the user with options for seeking additional information via certain key words such as HELP or WHAT. However, such a strategy is generally more appropriate to the development of large-scale interactive systems as opposed to singular interactive programs.

Problems of the third category above are, in addition to being very common, especially insidious. Program execution will often continue in spite of an input error; and furthermore, the student may never realize it, ultimately interpreting the final output as if his or her response had been processed as intended, as opposed to as interpreted, by the computer. Problems of this type will serve as the central theme for the remainder of this paper.

EVIDENCE OF THE PROBLEM

A study of user attitudes and input response patterns was conducted at Carthage College, a four-year liberal arts institution having a full-time day school enrollment of approximately 1100 students. User responses to several questions were solicited and recorded verbatim using an Apple II microcomputer and fifteen-inch color television. User behavior was also monitored and recorded via a video recording system.

The microcomputer system was put in an

empty classroom close to the daily student traffic between classes. The message "PLEASE REPLY" was displayed on the television screen using relatively large two-inch letters of randomly changing colors. Every 20 seconds the screen cleared and displayed the message, "MY NAME IS APPLE, WHAT IS YOUR FIRST NAME ?" in standard white characters on black background. This sequence was repeated continuously until any one of the Apple II keyboard keys was pressed. Following any single key response a maximum of 20 seconds was allowed for entry of any additional characters. If any 20-second period elapsed with no additional character input, or if the Return key was pressed, program execution continued with the next screen display--the assumption being that the user had completed entry of his or her name. It is important to note that this initial request for input did not include any specific instructions regarding format or operation of the keyboard, e.g., use of the Return key. The user's behavior and response pattern independent of such guidance was of primary interest.

The next screen display to appear was the message, "YOUR LAST NAME (OPTIONAL)." Following this display, program control was the same as that described above.

The next screen display was dependent upon whether or not the student had pressed the Return key after entering his or her first name. If Return had not been pressed the following message appeared: HELLO (name) ! IN THE FUTURE PLEASE PRESS THE (RETURN) KEY AFTER TYPING YOUR RESPONSE." If Return had been pressed, the following alternate message appeared: HELLO (name) ! I SEE THAT YOU ALREADY KNOW ABOUT THE (RETURN) KEY. These response-specific messages enhanced the feeling of participation by the student, as well as provided the student with feedback about the Return key.

After this initial interactive sequence, a series of five questions was displayed one at a time with appropriate pauses for student response. Each question except the last one was of the multiple option type, and below each such question the following message was displayed: "ENTER A SINGLE LETTER THEN PRESS THE (RETURN) KEY." ("LETTER" here refers to the letter preceding the response option selected by the student.) This message provides specific instructions regarding input format and keyboard control so that user response patterns under such conditions could be evaluated. However, any sequence of characters was recorded as the student's response pattern, and program execution always continued with the next question as long as the Return key was pressed. Using this technique it was possible to keep an accurate record of all input errors, including typographical and format errors, and at the same time avoid problems of program flow. If 20 seconds elapsed without the Return key being pressed, program control shifted to the initial PLEASE REPLY message assuming that the student had terminated interaction before answering all of the questions. This assumption was valid in all cases as evidenced by the video tape records of student interaction behavior. Each of the five questions and the corresponding response options are listed in Figure 1.

```
1.  IS THIS THE FIRST TIME THAT YOU'VE INTERACTED WITH A COMPUTER?

    A.  YES    B.  NO
-----------------------------------------------------------------------
2.  WHERE DID YOU HAVE YOUR FIRST COMPUTER EXPERIENCE?

    A.  IN THIS ROOM (PREVIOUSLY)
    B.  IN HIGHSCHOOL
    C.  AT HOME
    D.  AT A FRIEND'S HOUSE
    E.  AT A RELATIVE'S HOUSE
    F.  IN A CARTHAGE COURSE
    G.  NONE OF THE ABOVE
-----------------------------------------------------------------------
3.  WHICH CATEGORY BEST DESCRIBES YOUR FEELINGS ABOUT COMPUTERS?

    A.  VERY NEGATIVE
    B.  NEGATIVE
    C.  NEUTRAL
    D.  POSITIVE
    E.  VERY POSITIVE
-----------------------------------------------------------------------
4.  WHICH OF THE FOLLOWING CONVERSATION SPEEDS DO YOU PREFER?

    A.  VERY SLOW..........LIKE THIS (display speed corresponding)
    B.  SLOW...............LIKE THIS    "        "        "
    C.  MODERATE...........LIKE THIS    "        "        "
    D.  FAST...............LIKE THIS    "        "        "
    E.  VERY FAST..........LIKE THIS    "        "        "
-----------------------------------------------------------------------
5.  HOW MANY MINUTES WOULD YOU LIKE TO SPEND...SOME DAY IN THE
    FUTURE...INTERACTING WITH THE APPLE II AS YOU HAVE BEEN DOING?

    ENTER NUMBER OF MINUTES   THEN PRESS RETURN KEY
-----------------------------------------------------------------------
NOTE:  The statement:  ENTER SINGLE LETTER THEN PRESS RETURN KEY
       appeared after each of the first four questions.
```

FIGURE 1 Listing of questions used in the interactive sequence.

RESULTS

During a two-week period we collected 78 student response records. Eight of these response records were eliminated from analysis because they represented the second response record for a given student, involved an irregularity in the response sequence, e.g., two students responding alternately within the same session, or included an excessive series of uninterpretable or obviously insincere responses. The decision to eliminate a given response record was based upon the actual response record itself and the video tape record of student behavior during the interaction.

The following summary of response record data will be presented in conjunction with questions which are pertinent to the central problem of user input errors.

1. What percentage of students will respond appropriately to a request for personal information such as the person's first name? Sixty-seven percent of the 70 respondents made what was judged to be a sincere entry of their first name. The remaining 33% entered joke names, e.g., Jimmy Carter, or a meaningless series of characters.

2. What percentage of student respondents will press the Return key if not required or instructed to do so during the initial stages of interaction? Sixty-three percent of all respondents did press the Return key after entering their name, even though not instructed or required to do so. Interestingly, about half (48%) of these individuals were first-time users who were not expected to know about the general use of the Return key. Furthermore, of the 37% of all respondents who did not press the Return key, over one third (35%) were not first-time users and were expected to know about the general use of it.

3. What percentage of students will make typographical or format errors which would, given a poorly written program, result in continued but inappropriate program execution or ultimate termination of program interaction? Nineteen percent of all respondents made at least one typographical error during response entry which, given a poorly written program, would have resulted in continued but inappropriate program execution. Forty-six percent of all respondents made at least one format error which would have had the same consequence. In total, 50% of all respondents made either at least one typographical error or at least one format error.

4. Are students who are first-time users more likely to make input errors? Thirty percent of all first-time users made at least one typographical error while only 4% of the remaining respondents made at least one typographical error. This difference in proportions was found to be significant using the Chi Square test for independence of categories ($X^2=10.41$, df=1, $p<.01$). However, a significant difference between the percentage of first-time users making at least one format error (52%) and the percentage of the remaining respondents, i.e., not first-time users, making at least one format error (39%) was not found ($X^2=.85$, df=1, $p>.05$).

5. Are student attitudes regarding computers related to input error response patterns? There appears to be no significant relationship between these variables. Twenty-five percent of those who expressed a very negative attitude toward computers and 18% of those who expressed a very positive attitude toward computers made one or more typographical errors. This difference in proportions was not significant ($X^2=.25$, df=1, $p>.05$). Seventy-five percent of those who expressed a very negative attitude and 39% of those who expressed very positive attitudes made one or more format errors. This difference was not significant ($X^2=2.49$, df=1, $p>.05$).

DISCUSSION

The following general statements are consistent with the data obtained. User input frequently does not conform, either in form or intention, to the input requirements of many interactive programs. Furthermore, the occurence of such errors is not clearly related to the experience of the user. General guidelines such as those suggested by Hansen (1971) who states as his first principle of user engineering: "Know the user"; or that suggested by Pew and Rollings (1975) who state as their first principle of designing interactive systems: "Know the user population," are not sufficient to avoid interactive problems. Given the types and frequency of errors made, the suggestion made by Wasserman (1973), i.e., "Provide a program action for every possible type of user input," appears to be a directly relevant strategy for avoiding the problem of user-software interaction breakdown. This approach will serve as the focus for the following examples and recommendations.

RECOMMENDATIONS

Figure 2 illustrates a segment of a larger program. Assuming that the user responds with either "RR" or "RNR," this program segment will function properly. However, now assume for the sake of argument that a user who intends to implement the RANDOM WITH REPLACEMENT routine responds incorrectly, but not illogically, with "RWR." In such a case the requirements of statement 300 are such that any response other than "RR," including responses such as "(RR)," "R-R," or "RWR," none of which are illogical or highly improbable, will result in a continuation of program execution at statement 320, i.e., the beginning of the RANDOM WITH NO REPLACEMENT routine. As mentioned earlier, this type of problem is both common and insidious. Under such conditions the program continues execution using a method which the user may not intend and may not even be aware of. In the present example, the user may very well interpret the final output under the assumption that the RANDOM WITH REPLACEMENT routine was used.

```
.
.
.
200  PRINT  "WHICH METHOD OF NUMBER GENERATION SHOULD BE USED?"
220  PRINT  "RANDOM WITH REPLACEMENT  (RR)"
240  PRINT  "RANDOM WITH NO REPLACEMENT  (RNR)"
260  PRINT  "WHICH";
280  INPUT M$
300  IF M$ = "RR" THEN 540
320  REMARK..............BEGIN  RANDOM WITH NO REPLACEMENT ROUTINE
.
.
.
500  REMARK..............END    RANDOM WITH NO REPLACEMENT ROUTINE
520  GO TO 720
540  REMARK..............BEGIN  RANDOM WITH REPLACEMENT ROUTINE
.
.
.
700  REMARK..............END    RANDOM WITH REPLACEMENT ROUTINE
720  REMARK..................CONTINUE EXECUTION OF MAIN PROGRAM
```

FIGURE 2 First example of program segment having sequence flaw.

Figure 3 illustrates a revision of this program segment which avoids the problems discussed above. The addition of statements 305, 310, and 315 guarantee that no responses other than "RR" or "RNR" will be accepted as input. Consider what would happen even if statement 310 were not included in the revision. The user would be confronted with question mark after question mark while continuing to respond perhaps with "RWR" instead of "RR."

```
.
.
.
300  IF M$ = "RR" THEN 540

305  IF M$ = "RNR" THEN 320

310  PRINT "TYPE EITHER  RR    OR    RNR"

315  GO TO 280
.
.
.
```

FIGURE 3 Program statements for correction of sequence flaw in first example.

It is often argued that the level of user sophistication should dictate the amount of effort expended in writing or revising programs to avoid sequencing difficulties such as those illustrated above. However, even the most sophisticated user is capable of responding incorrectly. In some cases higher levels of user sophistication actually increase the probability of inappropriate input. Consider, for example, the program segment illustrated in Figure 4. If an individual had some degree of computer sophistication and was confronted with the question illustrated, he or she might quickly type "BIT" as a response. As a result, program execution would continue as follows: 1380...1420...1460...1520...1540. Consequently, the variable X(I) would be assigned a value of "4," and it would appear as though the user's intended answer was "BID."

```
.
.
.
1180  PRINT  "QUESTION #1"
1200  PRINT  "WHICH OF THE FOLLOWING ACRONYMS"
1220  PRINT  "IS USED TO REPRESENT THE CONCEPT"
1240  PRINT  "OF BINARY DIGIT:"
1260  PRINT  "                             A.   BYTE"
1280  PRINT  "                             B.   BIT"
1300  PRINT  "                             C.   BITE"
1320  PRINT  "                             D.   BID"
1340  PRINT  "  WHICH";
1360  INPUT R$
1380  IF R$ NOT EQUAL "A" THEN 1420
1400  X(I)=1
1420  IF R$ NOT EQUAL "B" THEN 1460
1440  X(I)=2
1460  IF R$ NOT EQUAL "C" THEN 1520
1480  X(I)=3
1500  GO TO 1540
1520  X(I)=4
1540  REMARK....................CONTINUE EXECUTION OF PROGRAM
```

FIGURE 4 Second example of program segment having sequence flaw.

Figure 5 illustrates a revision of this program segment which corrects for this problem. With this revision a user response of "BIT" would not be permitted and, as important, the user would be given specific information regarding the proper response format.

```
1380  IF R$="A" THEN 1500
1400  IF R$="B" THEN 1540
1420  IF R$="C" THEN 1580
1440  IF R$="D" THEN 1620
1460  PRINT  "ENTER EITHER  A   B   C   OR   D"
1480  GO TO 1360
1500  X(I)=1
1520  GO TO 1640
1540  X(I)=2
1560  GO TO 1640
1580  X(I)=3
1600  GO TO 1640
1620  X(I)=4
1640  REMARK....................CONTINUE EXECUTION OF PROGRAM
```

FIGURE 5 Program statements for correction of sequence flaw in second example.

SUMMARY AND CONCLUSION

Several years of experience in an instructional, time-sharing environment where interactive programs have been utilized have resulted in an accumulation of anecdotal evidence regarding the pervasiveness of problems which result from user input errors. The data presented in the present paper are consistent with this experience in that they demonstrate the frequency of such errors. These errors, when they occur in the context of interactive programs which have flaws of the type illustrated, are at best likely to result in confusion, frustration, and possible termination of a given interactive session. At worst the result may be an undetected error on the basis of which the student may draw incorrect conclusions.

The given examples of program segments which have sequencing flaws are intended to be representative of common limitations which are likely to be present in many interactive programs. These flaws can be easily detected by intentionally entering incorrectly typed, illogically stated, or improperly formatted responses while interacting with suspect programs. Resulting difficulties should be self-explanatory, and a simple scan of the program listing will often lead to an understanding of the changes, frequently minor, which could eliminate the problem.

I hope this paper has contributed to an increased awareness of and sensitivity to the problem discussed; and as important, to the understanding that programs can be written or easily modified so as to avoid this problem.

REFERENCES

Cheriton, D.R. Man-machine interface design for time-sharing systems. Proceedings of the ACM National Conference, 1976, 362-380.

Gaines, B.R., & Facey, P.V. Some experience in interactive system development and application. Proceedings of the IEEE, 1975, 63, 894-911.

Hansen, W.J. User engineering principles for interactive systems. Proceedings of the Fall Joint Computer Conference, AFIPS Press, Montvale, New Jersey, 1971, 39, 523-532.

Pew, R.W., & Rollins, A.M. Dialog specification procedure. Bolt Beranek and Newman, Report No. 3129, Revised Edition, Cambridge, Mass., 02138, 1975.

Sheiderman, B. Software psychology: Human factors in computer and information systems. Cambridge, Mass.: Winthrop Inc., 1980.

Wasserman, T. The design of idiot-proof interactive systems. Proceedings of the National Computer Conference, AFIPS Press, Montvale, New Jersey, 1973, 42.

Weinberg, G.M. The psychology of computer programming. New York: Van Nostrand Reinhold Co., 1971.

TRANSPORTABILITY OF INSTRUCTIONAL COMPUTER PROGRAMS: ISSUES AND EXAMPLES

Eugene A. Herman
Grinnell College

INTRODUCTION

Good instructional computer programs are difficult to develop and thus worth sharing. So we must consider transportability issues. I consider two very different methods for achieving (different degrees of) transportability: the easy method of writing programs for a single widely used computer and the hard method of writing machine-independent programs. I examine the problems associated with each method and discuss partial solutions to the harder problems. Finally, I describe in some detail how I have approached the problems of transportability in my current project and the effect this has had on my work.

In part, this paper is a call for our professional organizations to support specific measures that will help us share computer-based educational materials and our knowledge.

WHY IS TRANSPORTABILITY AN ISSUE?

As recently as 1972, it was still common to find overly optimistic predictions, such as this one in a major study done for the Carnegie Commission on Higher Education [6, p. 92]:

> Manufacturers of equipment for uses in teaching and learning at colleges and universities will have made a greater effort to adapt their designs so that compatible instructional components can be produced for use on a wide variety of makes and models.

In fact, there is today a much greater variety of makes and models of incompatible computer hardware and languages than in 1972. As those who fund computer-based educational projects recognize, these incompatibilities lead to high cost in the production and dissemination of materials and inhibit their use and development. For example, Andrew Molnar of the National Science Foundation recently wrote:[15]

> the lack of good computer-based educational materials is the major obstacle to the widespread use of computers and...this need is only partially being offset by local efforts--but at costs which are prohibitive and unnecessary when viewed from a national perspective. [emphases mine]

Also, Richard Hooper, Director of Great Britain's National Development Programme in Computer Assisted Learning has written [10, p. 5]:

> A major problem associated with the growth of computer assisted and computer managed learning will, in the forseeable future, remain cost....Indeed, software costs will increase, because software is and will remain labour-intensive....The claims regularly made that author languages will reduce software costs because they are 'easy to use' seem...much overstated.

ADVANTAGES OF TRANSPORTABILITY

We are, thus, well advised to remind ourselves of the advantages gained by developing materials that are highly transportable:

1) Funding agencies with a national perspective will look more favorably on our proposals.
2) Our materials will reach a far greater audience and thus the development cost per user will be much lower.
3) Our materials will benefit a much greater number of students and teachers.
4) We improve the quality of our materials when we can solicit the views of a greater number of students,

teachers, authors, and evaluators.

5) Our materials will benefit authors at other institutions who will see our work and learn from our triumphs and mistakes.

MICROCOMPUTERS AND THE EASY METHOD

In the last two or three years, however, the hard problems of making instructional computer programs transportable have been sidestepped by designing them for a single widely used standalone computer. For example, if a teacher writes a program for an Apple II micro-computer, then hundreds of other teachers can probably run the program. And those who cannot may be willing to purchase the computer, especially if the programs are very good and the hardware is cheap. The five advantages of transportability listed above seem to be largely achieved by this method and without any added effort. But this method has shortcomings which every author ought to weigh before deciding which method to choose for achieving transportability.

First, microcomputers are likely to have shorter lives than software (which needs to be lasting to justify its cost). They are not constructed to last longer than a terminal, and their technology is changing so rapidly that current hardware will soon be obsolete (perhaps by 1982 when the second generation of microcomputers is due). Because they are so new and changing so rapidly, we must also consider the likelihood that our programs will need many changes before they can run on the machines that replace them. The company that makes our computer may go out of business; or we may judge another company's new computer to be superior; or the next generation of a company's computer may have some features incompatible with its present one (which is especially likely in its way of generating graphics).

Second, microcomputers are designed primarily for the very large personal computing market, and their manufacturers are not particularly sensitive to the needs of educators. For example, the graphics capabilities of some micros make them seem particularly attractive for use in education, but their graphics hardware and most of their software are designed for games. For many applications in education, the resolution of the screen is too low, the speed of computation is too slow (especially when, as is typical, the language is an interpreted Basic), and the graphics software is too awkward or weak. Another drawback of personal computers is the inadequacy of their authoring aids. Authors who develop programs on larger systems have learned the value of large utility libraries, including powerful editing and debugging facilities, good graphics packages, and code-writing programs for mechanical sections of code. Such aids are being developed only slowly, and often inadequately, for microcomputers. Another kind of aid we ought not overlook is that provided by computer center staff. Often, computer centers refuse to consider microcomputers as part of their equipment, and we find it difficult to get professional help with our technical problems.

Finally, in order to compensate for the inadequacies of microcomputers, experienced authors often select a more uncommon machine or uncommon set of options.[3] For example, the Pascal operating system on a Terak or on an Apple II Plus or Apple III gives the author a good screen editor, a powerful and fast language, and a graphics capability with some superior qualities. But then the author has given up transportability, since far fewer people will have such hardware. Some authors improve the performance of their programs by exploiting idiosyncracies of their hardware. For example, there are frequent articles on tricks to speed up graphics output, but the methods used are unlikely to carry over to the next generation of machines.[13] Furthermore, an author who takes into account the idiosyncracies of a specific computer risks making even the design of his or her instructional computer programs dependent on the hardware, thus inhibiting future development of the design.

THE PROBLEMS OF THE HARD METHOD

On the other hand, every author should also weigh the problems associated with designing machine-independent computer programs. To sharpen our appreciation of these problems, let us consider them in a context that exacerbates them. Specifically, let us assume that:

a) We want to transport a collection of large, complex, instructional programs that call for computer graphics routines and file manipulations.

b) We want many other educational institutions to be able to install and run our programs on very different equipment from our own and with little difficulty. Although it may take time to modify our programs, it should be a mechanical process that leads to no significant degrading of their performance.

The following problems are then

quite significant. I will discuss partial solutions, but I do not expect the problems to disappear. I offer the problems and solutions as agenda items for us to bring to our professional organizations. For, by acting in concert, we can help reduce these problems and thereby come closer to achieving the advantages of transportability.

1) High-level general-purpose languages commonly used by authors, such as Basic, Fortran, and Pascal, exist in a bewildering variety of dialects. A program written in one dialect may need extensive rewriting before it can be run in another computer's dialect. Authoring languages, such as Coursewriter, Tutor, Common Pilot, and Decal, have not spread to a wide variety of computers.
2) Transporting programs that use computer graphics is especially difficult. The variety of graphics hardware is great; the "standard" versions of Basic, Fortran, and Pascal contain no graphics commands; and there are no standards for graphics software.
3) Other frequently desired capabilities, such as file manipulation and cursor control, are also system- and device-dependent.
4) Developing large, complex, high-quality, transportable instructional programs usually entails several laborious rewrites of the programs. Much of this extra effort may be blamed on problems associated with making the program transportable.
5) It is only worthwile expending extra effort to make instructional computer programs transportable if one can be assured of their high quality. But quality is hard to achieve and hard to measure.
6) It is difficult to find reliable information to help us cope with the above problems.

SOLVABLE PROBLEMS

Before discussing solutions to these problems, I want to briefly consider another class of problems which I regard as far more solvable. In order to make effective use of good instructional computer programs, we must have access to hardware and software with certain minimal capabilities. Our professional organizations could prepare a checklist of these minimal capabilities, which would give us added ammunition in battles to bring our computer facilities up to par. Also, authors could work with more confidence, knowing that many institutions would have certain minimal capabilities that are crucial to effective use of their programs. Although many others have offered such checklists,[3,8] I will summarize them and add items of my own.

We should have microcomputers or CRT terminals with screens that hold 24 lines of 80 characters each, handle upper and lowercase characters in several character sets, and have flexible cursor controls. We should have graphics microcomputers or CRTs that have a resolution of at least 500 by 500, can compute and display 100 vectors within one second, and have at least one graphic input tool. Our computer should be able to perform some standard manipulations on sequential and random access files. It should be able to run large programs in standard versions of the most common high-level programming languages. And, it should run standard graphics software. Some of these capabilities can be acquired easily, while some still require definition (especially) those with the adjective "standard").

PARTIAL SOLUTIONS TO THE HARDER PROBLEMS

Problems 1-6, on the other hand, may never be solved in any completely satisfactory way. But we can minimize them, especially if we can put the weight of our professional organizations behind some of the partial solutions. In the following paragraphs, I consider each of the six problems in turn and suggest partial solutions. In the final section I discuss the solutions I have adopted for the ** YOU SOLVED IT! ** project and the effect these solutions are having on my work.

Only Basic, Fortran, and, possibly, Pascal are sufficiently widespread and general purpose to be suitable as transportable authoring languages. Dialects of Basic are extremely varied, and the intersection of all these dialects is an extremely weak language. ANSI (the American National Standards Institute) published a standard called Minimal Basic in 1978.[14] But since adherance to standards progresses slowly, a safer approach is the use so-called "Level 0 Basic" defined in a booklet distributed by CONDUIT.[2] Fortunately, ANSI has established committee X3J2 on Basic to develop a far richer standard. Fortran, on the other hand, has been standardized by ANSI since 1966. Although the standard was updated in 1977, that version is still not widespread. In fact, since some implementations of Fortran do not even meet the 1966 standard, it is safer to use a subset which has been carefully defined by Bell Laboratories. Bell distributes a verifying program

PFORT which, when fed a Fortran program, will identify all the ways in which the program fails to meet its more stringent standards.[17] Pascal is not yet nearly as widespread as Basic or Fortran but is gaining acceptance rapidly. Unfortunately, as it spreads, dialects multiply. Even so-called "Standard Pascal" is not free from significant transportability problems.[11] ANSI committee X3J9 is working on an official standard.

We can only hope to transport programs that use computer graphics if there are standards for graphics. Fortunately, two ANSI committees are working on grapics standards. Committee X3J2 (on Basic) is considering a standards proposal that includes graphics commands originating at Dartmouth College. Committee X3H3, however, is charged with the development of a language-independent standard. Although it is working from a 1979 proposal by SIGGRAPH,[18] X3H3 expects the graphics standards produced by the two committees to be compatible. Several individuals and groups have produced implementations of SIGGRAPH's 1979 proposal, but only the one I have developed seems to be a likely candidate for use in education.[7] For easy but limited transportability, authors might consider one of the device-dependent graphics packages that runs on a single widely-used device. There is Tektronix' PLOT 10, the Calcomp software, and Apple's new Applegraphics package for its Pascal operating system.

The most nearly transportable way of handling system-dependent and device-dependent functions, such as file manipulation and cursor control, is to relegate them to small, well-documented subroutines that can be altered easily at other sites. But authors need to know which functions are generally available, to avoid writing subroutines whose functions can be reproduced at only a few sites. For this, we need the aid of our professional organizations to recommend minimal capabilities for computers in education.

The key to efficient development of large, complex, instructional programs is planning. The author must not only think out an instructional design very carefully, but must plan how to avoid transportability problems. The style of coding can have a great effect on the ease with which programs can be written and rewritten. Structured programming techniques and good documentation (see 12) are important since they make the code manageable and understandable, thus simplifying debugging and rewriting. Authoring aids such as good editing and debugging facilities and automatic code-writers for mechanical sections of code speed up the work of programmers.

Others have written extensively on how to structure a project to make high quality easier to achieve and test, so I will only summarize their recommendations.[4,5] A team of two or three instructors working together can try out pedagogical ideas on one another. A professional instructional designer can help them implement their ideas. A computer scientist can help guide the production process. And an educational psychologist can design the testing and evaluation procedures. Evaluation should begin at the earliest stages in an informal manner by trying out design ideas on students, faculty, and others. First versions of programs should be tried out in classes and be examined by other authors and faculty. Final versions should undergo rigorous testing and evaluation.

Although reliable information about the above problems is difficult to find, there are many useful sources. CONDUIT is a national organization (with headquarters at the University of Iowa) which reviews and distributes instructional computer programs to colleges and universities. Transportability is a major concern of theirs and they distribute some useful booklets.[2,16] The American National Standards Institute publishes standards that make transportability possible. Useful information can also be found in journals and at conferences such as those supported by SIGCUE, ADICS, and AEDS and those supported by the disciplines in which we teach. Access to a large library and a good travel budget are important.

THE ** YOU SOLVED IT! ** PROJECT

Although National Science Foundation funding for my project did not start until 1979, I began related work in 1977. My project has been to develop and evaluate instructional computer programs for use in pre-calculus college mathematics. The programs combine elements of drill and practice, interactive testing, and dialogue. They are large and complex; they use sequential files; and some use computer graphics.

I rejected Basic since Minimal Basic is a weak language and since interpreted Basic is usually too slow for graphics and even too slow for some of our elaborate answer analyses. I regarded Pascal as insufficiently widespread and so I used 1966 Fortran. My programs will also pass the more restrictive PFORT verifier. But my information on transportable Fortran was inadequate, and we wasted time rewriting some non-transportable code.

I regard transporting of computer graphics programs to be the most challenging and interesting problem. I have designed two successive language-independent and device-independent graphics packages based on SIGGRAPH's proposal.[18] We have implemented these in Basic and Fortran and have used them extensively, but mostly locally. Some of our latest subroutines are for graphing in three-space and manipulating software-generated text. I am currently waiting to hear if the National Science Foundation will fund my proposal to design and produce highly transportable implementations of the coming graphics standards for use in education. The effort we have put into developing graphics has been substantial, but the benefits of our work could be widespread.

In addition to graphics functions, we perform all file manipulation, cursor control, and other common functions in small, well-documented utility subroutines. For example, UCURSOR(I,J) moves the cursor I lines up and J columns right, URCORD records information about the current user in a sequential file, UINPUT prompts the user and partially analyzes the response, and UVALUE computes the value of a numeric string. Those subroutines which are system- or device-dependent then need only be rewritten once at another site for them to work on all the ** YOU SOLVED IT! ** programs. We have almost 25 utility subroutines, not counting those in our graphics package. We need more study, however, to understand what constitutes a good utilities package.

Careful structuring of subroutines has been our key to efficient writing and rewriting of programs. Although Fortran is not as well structured as Pascal, for example, it does have separately compilable subroutines. I impose even more structure on it by requiring programmers to adhere to additional rules. All code is carefully constructed from the top down. No program or subroutine can be more than two pages long (including comments) and frequently used subroutines are usually only a few lines long. The use of COMMON blocks, DIMENSION statements, and type declarations is carefully controlled. All code is handwritten and hand tested before being entered in the computer. Many subroutines are tested individually in the computer. A typical program has about 2000 lines and is broken into 20-30 subroutines plus the utility subroutines. We use a powerful screen editor, VTEDIT, for writing and rewriting code. We also have two automatic code-writers to handle the very tedious parts of Fortran coding. The FORMAT statement generator is a Basic-Plus program which takes a file representing a full screen of alphanumeric output (created easily using VTEDIT) and from it produces the necessary FORMAT statements. The DATA statement generator is a Basic-Plus program which takes a file of alphanumeric strings and produces a set of DATA statements in which each string is loaded into an array, one character in each array element. Those arrays can then be easily manipulated as strings. We wasted a lot of effort before settling on our present structuring methods and our code-writing programs. But I am finally satisfied that we have an efficient system for producing clear and reliable code.

The most important step I take to insure the quality of my work is to ask myself what I most want my students to learn and what effective contribution the computer can make that cannot be equalled by other means. Since there are no other authors nearby for me to consult, I try out my ideas on all the students, faculty, and friends who will listen. Then I use first versions of the programs in my own classes and persuade other faculty to use them in theirs. I add questions to end-of-course evaluation forms to solicit the opinions of all students who use the programs. Also, CONDUIT has solicited some preliminary reviews of my programs and colleagues at other institutions have tried them out. I am presently at the final stage for some of the programs--a formal experiment to evaluate them. To insure that this is done well, I am working with an educational psychologist and a statistition. I will report on the results of this experiment in my delivery of this paper to the conference.

Much of the information that guided my work in the beginning came from CONDUIT. Their information has been reliable and substantial, but certainly not complete. I have also gleaned useful information related to transportability from NECC and CCUC and from computer-related journals and magazines too numerous to mention. A lengthy sabbatical has given me the time to do much of this reading, traveling, and learning.

REFERENCES

1. Ashby, Neil. Development of Problem-Solving Skills in Physics (Electrostatics). National Science Foundation Grant No. 78-14618.
2. BASIC Guide. CONDUIT. 1978.

3. Bork, Alfred. Machines for computer-assisted learning. Educational Technology, vol. 18, 17-20 (April 1978).
4. Bork, Alfred. Educational Technology Center at the University of California, Irvine. IFIPS London Conference, September, 1979.
5. Brown, Bobby R. and Robert S. Ellinger. Evaluation: The Neglected Step in CAI Package Development. Pipeline, Vol. 4,4-10 (Fall, 1978).
6. The Carnegie Commission on Higher Education. The Fourth Revolution: Instructional Technology in Higher Education. McGraw-Hill. 1972.
7. Chappell, Gary. Implementations of the CORE. Computer Graphics, vol. 13, 260-278 (1980).
8. Gerhold, George. Computer series, 6:computer-aided instruction with microcomputers. J. of Chemical Education, vol. 57, 93-99 (1980).
9. Hall, Keith A. Review of Computer-Based Education: Research, Theory, and Development. Proceeding of National Educational Computing Conference 1979, 7-13.
10. Hooper, Richard. Computers in science education-an introduction. Computers and Education, vol. 2, 1-7 (1978).
11. Isaak, Jim. Avoiding Pascal's pitfalls. Mini-Micro Systems, vol. 13, 142-148 (December, 1980).
12. Legard, Henry F. and Louis J. Chmura, Jr. FORTRAN with Style: Programming Proverbs. Hayden Book Co. 1979.
13. Lubar, David. Apple lo-res shape tables. Creative Computing, vol. 7, 120-124 (January, 1981).
14. Minimal BASIC. ANSI document X.360-1978.
15. Molar, Andrew R. The next great crisis in American Education: computer literacy. AEDS Journal, vol. 12, 11-20 (February 1978).
16. Peters, Harold J. and James W. Johnson, Author's Guide. CONDUIT. 1978.
17. Ryder, B.G. The PFORT Verifier: User's Guide. Computing Science Technical Report No. 12. Bell Laboratories. 1973.
18. Status Report of the Graphics Standards Planning Committee. Computer Graphics, vol. 13 (1979).

The Next Step: The Use of the MicroSynergistic (Microcomputer Cluster) in Education

This project was supported by the
Buhl Foundation, Pittsburgh, PA

John W. Motto
Saint Vincent College
Latrobe, PA 15650

Abstract: The next step in the use of computers in the small undergraduate college is analyzed and found to be a group of personal microcomputers interfaced to the main academic computer. The design and implementation of a six-unit cluster is then described. The design features a simple approach using commercially available hardware.

INTRODUCTION

The wide-range potential advantages of computers in education has not yet been realized. Although the computer age is now going into its third decade and unbelievable strides have been made, the expected impact of computers in education has not yet arrived. This is especially true of the small undergraduate college. Appendix I analyzes the past, current, and future impact of computers on the small college. The conclusion from this analysis is that there are technological as well as psychological factors that stand as barriers to the potential of computers in education. To oversimplify these barriers they are: 1) The cost of computer equipment is too high to permit classroom use. What is really needed, and very possible in the foreseeable future, is computer availability such as found in home TV sets and typewriters and; 2) The number of educators with sufficient literacy in computer equipment is too small. This second factor is primarily a result of the first; no educator can start to understand how the computer can assist classroom instruction without hands-on experience with the computer.

I believe that just because the potential of computers in education has not been realized in a grand scale to date does not mean the potential is not there. What is needed is an objective analysis of how computers can best assist education today, and what is required to realize their potential in the future. That is to say, what is the next step?

Appendix I analyzes the past impact computer equipment has had on the small college and notes the importance of the historical price/performance ratio (industry trend of a 21 percent decrease in computer costs annually). The analysis also finds a trend in academic computer facilities to form a hierarchical parent-child structure. That is, a computer facility rarely stands alone, but is initially supported by a larger, often remote, computer. Due to the fundamental nature of this parent-child relationship, it is found that the next step is that of inter-connecting personal microcomputers into a group and then interfacing that group to the main academic computer facility. This is termed, for reasons to be discussed later, the microsynergistic group.

The next steps have immense impact on the delivery and mode of instruction in the small college. This paper, however, does not project beyond the next step, the development of the hardware and software necesary for easy access to the data-base resources of the main academic computer facility by the microsynergistic group; and the evaluation of the benefits and problems of this interface when used by the student, in conjunction with his own personal computer, and when used by the instructor in the classroom.

THE MICROSYNERGISTIC GROUP

The microsynergistic group is illustrated in figure I. The term "microsynergistic" has been coined from a biological term synergism which refers to the result of a group of "things" (microcomputers in this case) working together to produce a result which is greater than the sum of their individual capabilities. Historically computers, and humans for that matter, form hierarchical structures like families, companies, etc., where the resulting organization is much more capable than the sum of the individual parts working independently.

The operation of the microsynergistic group is as follows: The master micro has the software and hardware to access the main college academic computer and transfer programs and data to and from the group of microcomputers. This permits rapid downloading of courseware as required for computer-assisted instruction. It also permits the student to store programs and data on the main academic computer and make hardcopy printout on the line printer of the main academic computer.

These interfaces are not implemented easily. Although the 6502 microprocessor and the Apple (TM) system software were carefully chosen for input-output flexibility, there still were complex problems of timing, especially on inputing to the Nova (TM)[1] minicomputer which is operating in a multiuser swapping mode.

The advantages of the cost-effective personal microcomputers coupled with the data base and resources of the main academic computer center are great, and the required techniques have been developed. The consequences of the microsynergistic approach in education within the next few years are indeed exciting.

The MSG system has been developed by Saint Vincent College with support from the Buhl Foundation (Pittsburgh, PA) and is now in productive operation at Saint Vincent College. The MSG system is an important step in providing readily accessible, on-line instructional computer capabilities at lower cost than previously possible.

In addition to the lower costs, the processing speed of the individual microcomputers in the MSG system provides additional pedagogical features which were previously clearly out of the reach of the small college. These features are high resolution graphics, multi-color output, audio output, and animation. These features expand the dimensions of information presentation to the student by means of detailed diagrams and curves, color-keyed diagrams, sound prompting, and relative motion diagrams.

ECONOMICS OF THE MSG SYSTEM

The operating cost advantages of the MSG system can be clearly demonstrated through a comparison with commercially available time-sharing systems. These time-sharing systems cost $4.00 to $9.00 per terminal hour. The operating cost of the 5-unit MSG system on the Saint Vincent College Nova (TM) system has an estimated operating cost about 18¢ per terminal hour.[2] This cost is based upon 90¢ per terminal hour expenses per computer port on our system, divided by 5 for the 5 microcomputers sharing one port.

The equipment cost of the MSG system is also significantly lower, demonstrated by comparing the cost of adding a conventional terminal on the main academic computer system. The cost of an additional terminal is that of a terminal plus the cost of the communication port to interface the main computer. A plain text CRT terminal would cost $1,200, and the main computer port on the Saint Vincent College multiuser minicomputer system would cost $4,000. The total equipment cost to add one conventional terminal would therefore be $5,200. By comparison the 5 units in the MSG system share the $4,000 main computer port, resulting in a cost of only $800 per terminal. The equipment costs for each of the five terminals is $2,320, resulting in a toal cost of $3,120. This, compared to the cost of $5,200 for the conventional approach, demonstrates a significant cost reduction in academic computing.

The total cost of a projected classroom size MSG system with 16 terminals would be $250 for the prorated main computer port and $1,750 for the equipment cost, resulting in only $2,000 total. This equipment cost taken over a five-year life and at 4 hours per day would be 40¢ per hour. Adding the operating cost of main academic computer port, which would be prorated to only 6¢ per hour, results in a total cost of 46¢ per terminal hour.

This cost of 46¢ per student terminal hour compares favorably to other cost components of a college education. Tuition at Saint Vincent College cost $6.67 per class hour. A typical $15.00 textbook, used in a three credit hour course, costs 33¢ per class hour. These costs when compared to the 46¢ per student terminal hour on the MSG system illustrates the possibility of the MSG system having great impact on the use of computers in education in the very near future.

---

[1] Nova is a trademark of the Data General Corporation.

[2] Terminals on the MSG system are much more than terminals, they are stand-alone computers.

EDUCATIONAL COURSEWARE LIBRARY

An educational computer program library has been started on the Saint Vincent College computer system. Because the MSG system interfaces the main academic computer with its high speed, large capacity hard disk, it is relatively easy to maintain this library. This includes safe storage of programs and categorizing the programs by educational discipline, key words, alphabetically, etc. This library of MSG educational programs has the potential to assist other educational institutions which might become interested in the MSG system.

BASIC PROGRAM TRANSLATOR

While both the Saint Vincent NOVA system and the MSG system uses the Basic (an acronym standing for Beginners All-purpose Symbolic Instruction Code), there are unfortunate differences in the syntax of the two computer languages. A translator program has been written to convert the NOVA, Basic to MSG basic (Applesoft). At present this translator converts about 95% of the changes that need to be made to speed-up the conversion of programs written on the Nova system to run on the MSG system.

DESCRIPTION OF MSG SYSTEM

The MSG system was developed with the philosophy that we should use readily available commercial components if possible. To date we have been successful in achieving this objective. This philosophy improves the transportability of the MSG system to other educational institutions.

The current system uses the Apple Computer Company serial communications board. We have been successful in operating the MSG system up to 4800 baud which is about 480 characters per second. The individual microcomputers on the MSG system can operate as terminals on the Saint Vincent College Nova system which is operating in multiuser, swapping extended Basic. In this mode they can carry any of the functions given in Appendix II, such as running a program, creating a program, changing a program, and printing out the text of the program on the line printer. The individual microcomputers can also request that a program which has been stored on the Saint Vincent College Nova system be transferred to itself. After the program has been transferred, the microcomputer can run the program, change it, and when through, can link-up with the Nova system and save the program on the Saint Vincent College Nova systems disk storage. The MSG system also has several other features, such as local printer which can be shared by all the MSG terminals for hard copy of listings and runs of programs. The local printer can also print out a copy of the high resolution graphics. There is a color monitor (TV set) which all the MSG terminals can use to display program output in different colors. The individual microcomputers have black and white monitors (TV sets) to reduce costs. The audio amplifier of the color monitor can be shared by the individual terminals.

There is a group program input mode where all or any number of the individual terminals can receive the same program from the main academic computer system simultaneously. This mode would be required in a classroom environment.

SUMMARY

The MicroSynergistic Group has been analyzed to be the next step in the use of computers in education. A five-unit MSG system has been developed and in productive use in the Saint Vincent College Computer Center. This microcomputer group is a natural extension of computer networks and resource sharing.

The equipment costs of the MSG system are at least 40% less than the best alternative, a multiuser minicomputer system, and are projected to be 60% less for a 16-unit MSG system. The operating costs are 80% less for the five-unit system and 93% less for a projected 16-unit system. These figures are based upon current list prices of personal microcomputers which are sure to continue to decrease in cost in the near future.

These lower costs, with the additional dimensions of information presentation (i.e., graphics, color, sound, and animation) of the MSG system make the MSG system an important component in realizing the immense potential of computers in education.

APPENDIX I

THE ANALYSIS OF THE NEXT STEP

The most critical factor in a program such as the one proposed here is not finding a viable area to investigate and implement, but to find the best next step in the implementation of new technology into the educational process. This section will describe the analysis which was used to conclude that the microsynergistic group is the next step in realizing the immense potential of computers in education.

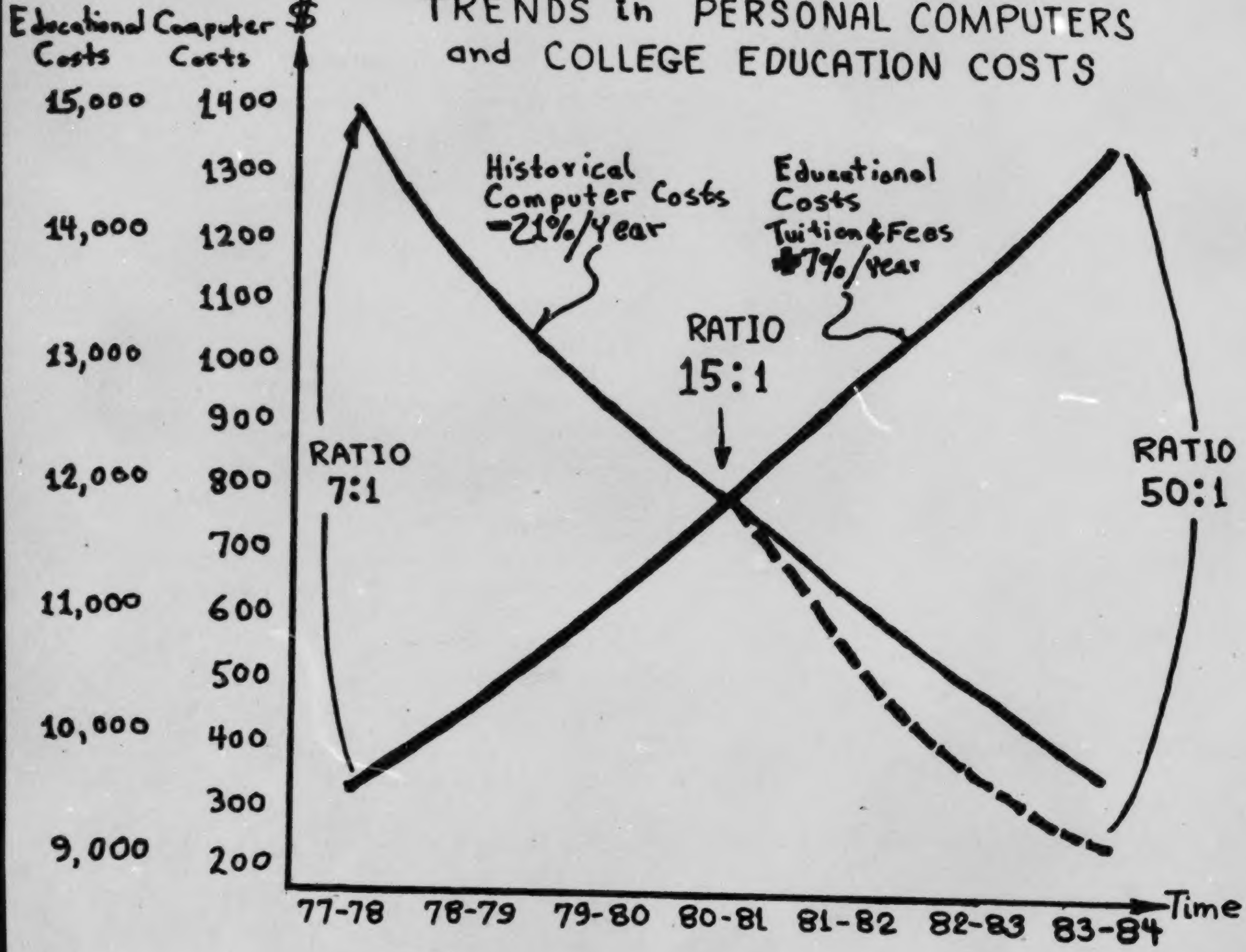
The historical decrease in price/performance ratio of computer equipment is 21 percent per year. This negative inflation factor is very significant in predicting when computer equipment will greatly influence the educational process. The cost versus time of computer equipment and education is illustrated in figure 1. Note that the ratio of the typical tuition and fee cost for a four-year undergraduate college education to the cost of a basic microcomputer was 7 to 1 just three years ago, but today is 15 to 1 and in the next three to four years will be as great as 50 to 1. With this ratio, the personal microcomputer will be considered much like a $300.00 scientific calculator was considered a few years ago for the science major, or a good typewriter for the non-science student. The dotted-line projected cost trend is greater than the historical 21 percent decrease in entering the commercial market. Here the volume and demands of that market will have an unprecedented effect on costs. Typically prices tend to decrease by a factor of 5 to 1 for computer-based equipment entering the commercial market. This is illustrated in figure 2. The conclusion that a $250.00 microcomputer will greatly influence the educational process is inescapable.

Now we must consider, not how personal computers will affect education four to five years from now, but how we can utilize the present $800 to $1,200 personal computers today to improve education. That is, what is the next step? First, there are indeed limitations to the present day microcomputer. The $800 to $1,200 personal computer has no hard copy output and very limited program storage capability. These accessories can be added but will tend to double the price, resulting in a marginal cost-benefit ratio.

If we analyze the historical use of computers in the small undergraduate college, a pattern occurs which clearly points the way as to how personal microcomputers can be used effectively. The trends of small college academic use of computers are diagramed in figure 3. This figure illustrates a typical computer configuration for various periods of time. In the period 1955-1965 computer facilities were typically located at large colleges and universities. Their operation was batch (i.e., card input and hard copy output). In the period 1965-1975 the computers were still located remotely from the small college but now had become multiuser time-sharing systems, and the small college could get access to these facilities over telephone lines. The cost was then, as it still is today, excessive; it was $5.00 to $12.00 per terminal hour and only limited use could be justified. In the 1975-1980 period, microcomputers with multiuser capability became available and permitted resident computer facilities on the campus of the small college. A typical system might consist of a 64 KB memory and a 10-megabyte hard disk which could service 5 to 9 on-line terminals. The system would still tend to rely on the remote computer time-share service such as could be accessed through EDUCOM for demanding processing or accessing large data-bases which could not be maintained or stored with its limited capacity. There was even the possibility of a terminal being located in the classroom, but typically students were, and still are, sent to the computer center to do computer-assisted instruction assignments.

This is where we are today, as indicated by the star in figure 3. The future, of course, can only be estimated, but we have already identified some trends. Computer systems tend to rely at least initially on a larger computer system, often remotely located. That is, computers tend to form networks to permit sharing of more costly functions. Thus we see projected in the 1980-1985 period the microsynergistic group. This is a group of microcomputers which appear as only one terminal on the main academic computer facility, and may be located in the computer center or in the classroom. The development and use of the microsynergistic group is the next step, the step which has the greatest potential of bringing computers into the classroom for the first time. However, it is interesting to analyze even further into the future the impact and mode of delivery computers will have in education. Note that in 1980-1985, the minicomputer has sufficient capability that it does not rely on the large remote computer for computing capability but continues to use it as a data base. Likewise, in the 1985-1990 period the college academic computer center disappears, except as a college data base, and the computer power really goes to the classroom. Also note the dotted lines which indicate that the students might well be carrying their personal computers back to the dormitory for homework.

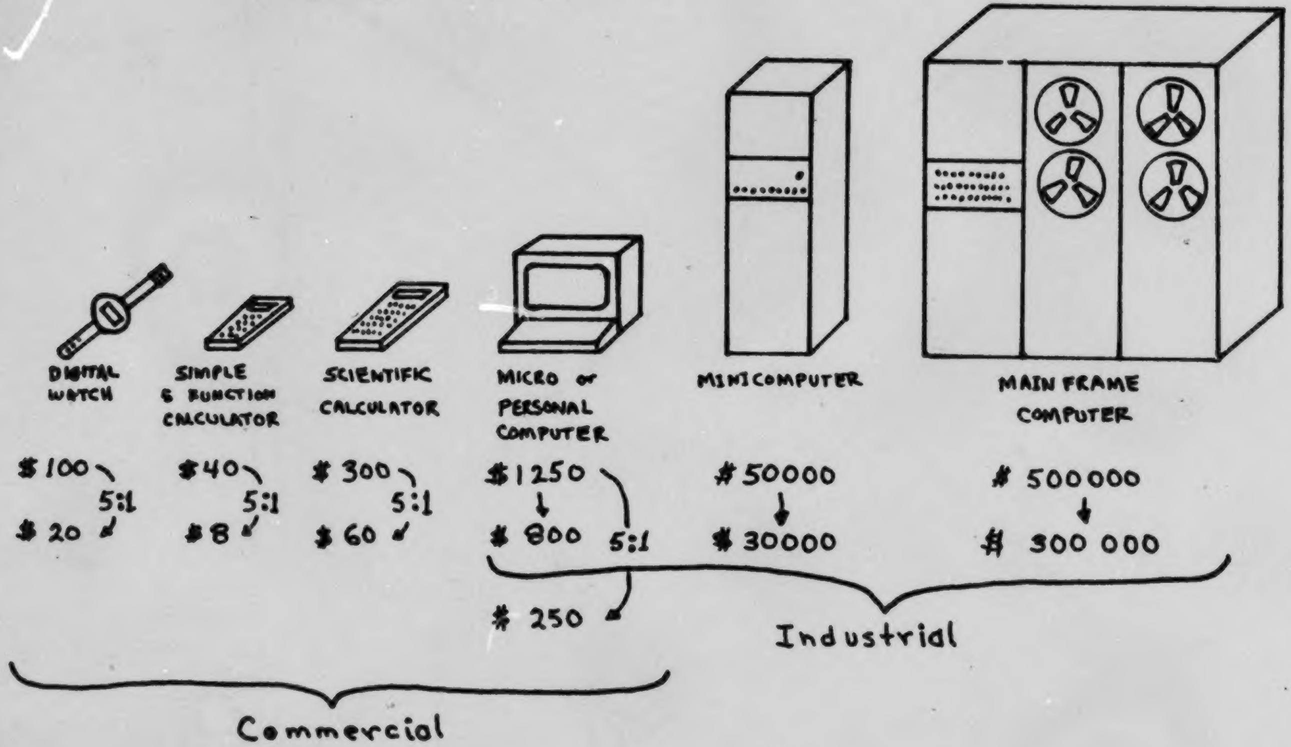
To continue projecting the trends developed, note what would happen in 1990-1995. Here the student would possibly just stay at home with his personal computer and connect to the small college instructor via the classroom computer which has the course information, tests, etc. The instructor will, no doubt, still want access to the college and remote data bases.



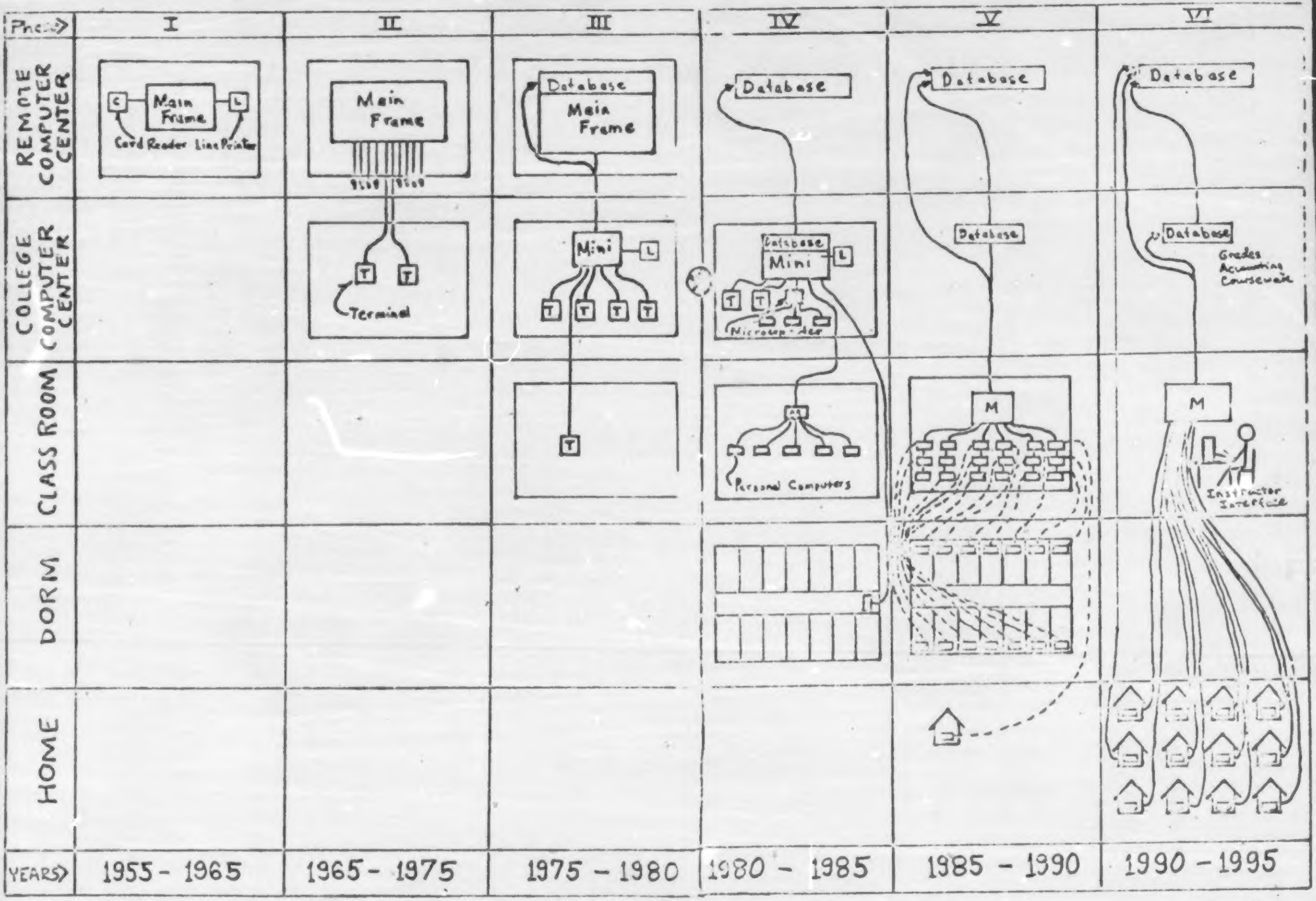
TRENDS in PERSONAL COMPUTERS and COLLEGE EDUCATION COSTS

COMPUTER TECHNOLOGY EQUIPMENT COST TRENDS

(Last 4-5 Years)


DIGITAL WATCH
$100
5:1
$20
SIMPLE 5 FUNCTION CALCULATOR
$40
5:1
$8
SCIENTIFIC CALCULATOR
$300
5:1
$60
MICRO or PERSONAL COMPUTER
$1250
$800
5:1
$250
MINICOMPUTER
$50000
$30000
MAIN FRAME COMPUTER
$500000
$300 000
Industrial
Commercial

TRENDS IN SMALL COLLEGE ACADEMIC COMPUTER FACILITIES


Phase
I
II
III
IV
V
VI
REMOTE COMPUTER CENTER
COLLEGE COMPUTER CENTER
CLASS ROOM
DORM
HOME
C
Main Frame
L
Card Reader
Line Printer
Main Frame
9600
9600
T
T
Terminal
Database
Main Frame
Mini
L
T
T
T
T
T
Database
Database
Mini
L
T
T
Microcomputer
Personal Computers
Database
Database
M
Database
Database
Grades
Accounting
Courseware
M
Instructor Interface
YEARS
1955 - 1965
1965 - 1975
1975 - 1980
1980 - 1985
1985 - 1990
1990 - 1995

APPENDIX II

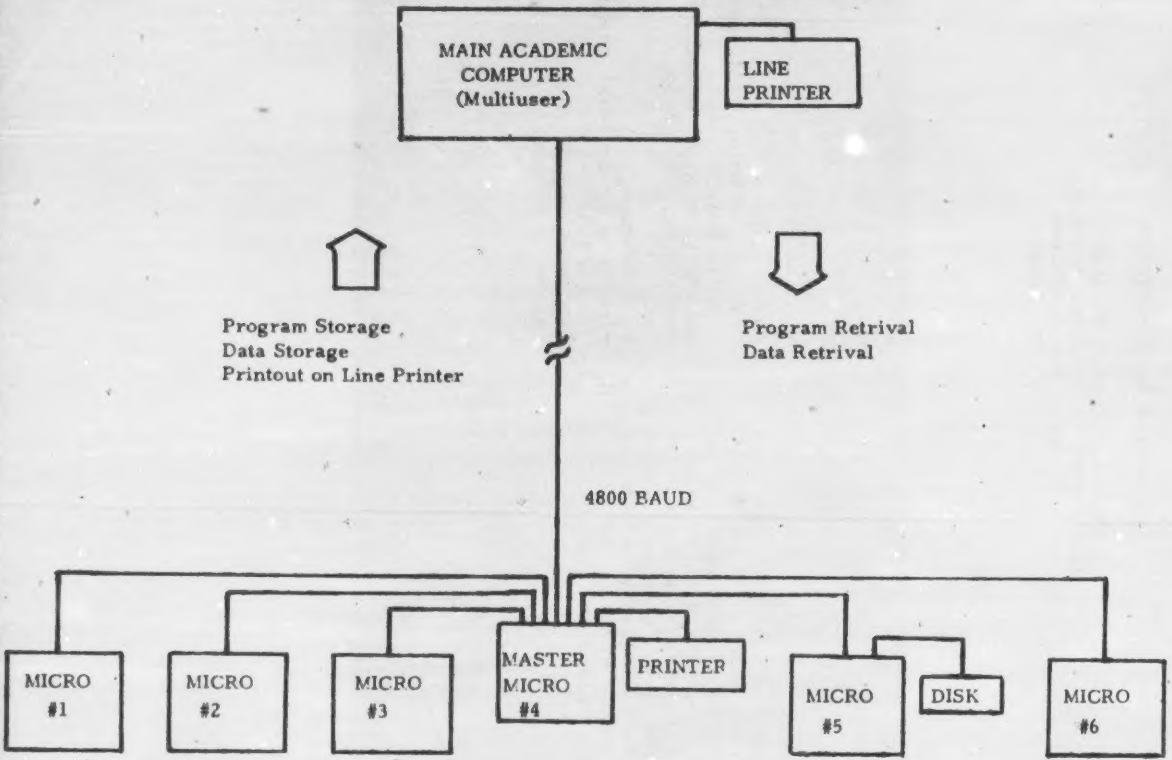
```
<%-%-%-%-%-%-%>  MicroSynergistic Group  Dual Nova Commands  <%-%-%-%-%-%-%-%>
                 =    =            =
```

Action (Terminal Mode)	Command	Comment
Enter Teminal Mode	IN#2(CR) ctrl(A) ctrl(F) (T)	This makes the Apple a Terminal on the Dual Nova System
Sign-on DUAL NOVA System	(ESC)then type in IDID(CR)	Apple must be in Terminal Mode
Delete Last Character	ctrl(H) Back Arrow	Will Delete last char.typed
Delete Current Line	ctrl(X)	Will Delete last line typed
Store Program on Disk	RUN "MSGLIST"(CR) enter PNAME.TX	Program in Apple will be Stored on NOVA Name PNAME.TX
Store Hex Code on Disk	RUN "MSGHEXDUMP"	enter PNAME.HX,Hex Start,End
Printout Prog. on Nova With Header & Pages	RUN "MSGPRINT"(CR) enter PNAME.TX	Prints latest copy on Nova
List Apple Prog that is stored on NOVA	RUN "LIST" enter PNAME.TX	Program will be Listed on Apple (Not Transfered)
Stop Listing	ctrl(S)	Will Halt Listing
Resume Listing	ctrl(S)	Listing will Resume
Terminate Listing	ctrl(Z)	To Stop listing
Assemble 6502 Code	RUN "6502ASM" enter PNAME.AX	Assembley Program will be Cross Assembled into P-.HX
Get Program from Disk	RUN "MSGENTER" enter PNAME.TX	Will Download Program into Apples CPU
Get Hex Code from Disk	RUN "MSGHEXLOAD"	enter PNAME.HX
Run Library Program	RUN "PNAME"(CR)	Runs Progran on NOVA
Run Prog(on Nova)	RUN(CR)	Prog must be Created or ENTERed into Workspace first
Rename Program	RENAME "OLDPN","NEWPN"	Changes Program Name on Disk from OLDPN to NEWPN
Delete Stored Program	DELETE "PNAME.TX"(CR)	Delete Program from Disk Be Sure to Delete Old Progs
Kill Action on Nova	(ESC)	A Short delay then * prompt
INFO on M S G System	RUN "INFO"(CR)	Info such as this; Code MSG
Edit Program or File	RUN "EDITNO"(CR)	PNAME & Line Nos Requested
Leave Terminal Mode	ctrl(E)	Be sure to Not leave Apple in Terminal mode

Action (Microcomputer Mode)	Command	Comment
HELP on MSG System (Info on Commands)	IN#2(CR) ctrl(A) ctrl(F) (H)	Will display Info on Apple Screen
List Apple Program (Output to Xerox)	IN#2(CR) ctrl(A) ctrl(F) (L)	Program will be Listed on Xerox Printer
Run Apple Program with (Output to Xerox)	IN#2(CR) ctrl(A) ctrl(F) (O)	Output of Program will be printed on Xerox
Printout High Resolution Screen on Xerox	IN#2(CR) ctrl(A) ctrl(F) (P)	High Resolution Screen will be Printed on Xerox
Leave O and T Modes	ctrl(E)	Do Not leave Apple in Communication Mode

FIGURE I

THE MICROSYNERGISTIC GROUP


MAIN ACADEMIC COMPUTER (Multiuser)
LINE PRINTER
Program Storage
Data Storage
Printout on Line Printer
Program Retrival
Data Retrival
4800 BAUD
MICRO #1
MICRO #2
MICRO #3
MASTER MICRO #4
PRINTER
MICRO #5
DISK
MICRO #6

FUNDING OF VIDEODISC PROJECTS

Sponsored by ACM-SIGCUE

Chaired by Alan Breitler
Department of Computer Sciences
U. S. Coast Guard Academy
New London, CT 06320

ABSTRACT:

This panel will discuss current and future funding possibilities for videodisc projects and research results obtained to date, sources of information and how to apply for project funding, the outlook for future funding, and a general discussion of the potential impact of videodisc use in instruction. A question and answer period will follow the panel discussion.

PARTICIPANTS:

Joseph Lipson
Division of Science Education Development
and Research
National Science Foundation
Washington, DC 20550

Kent Kehrberg
Minnesota Educational Computing Consortium
Videodisc Project
2520 Broadway Drive
St. Paul, MN 55113

COMPUTING CONCEPTS IN ELEMENTARY SCHOOLS

Sponsored by ACM-SIGCSE

Chaired by Margaret Christensen
Widener College
Chester, PA 19013

ABSTRACT:

This session will consider how much children of various ages can be expected to learn about computers, what hardware and software best enables this learning, and what textbooks, workbooks, etc., are needed to go along with the computer facilities. To develop a society of computer-literate adults, we need to provide children with an appropriate computer culture to grow up in. Children should learn to program computers and to use them as personal tools, from the earliest grades. The issue is not whether children CAN learn to do this, but whether adults are willing and able to provide an environment in which it can happen. The panelists will deal in part with the following kinds of questions.

What are appropriate concepts to be taught to elementary school children?

Should these be required or optional topics?

What can be taught about programming without algebra?

How can one teach a problem-solving approach to computing?

What computer languages are appropriate for use with children?

Should the study be entirely computerized or should there be workbooks to supplement the computer work?

Where in the curriculum should the computer concepts be placed?

What characteristics are required or recommended for hardware suitable for children to use?

When there is limited hardware, how should priorities be made?

What software is available to use to teach computing concepts to children?

Where are there successful programs in operation?

What are the important aspects of the work in which the panelists themselves have been engaged?

PARTICIPANTS:

Jacques LaFrance
Oral Roberts University
Tulsa, OK 74171

Arthur Luehrman
University of California
Berkeley, CA 94708

Robert Taylor
Teachers College
Columbia University
New York, NY 10027

Daniel Watt
LOGO Project
M.I.T.
Cambridge, MA 02139

Grayson Wheatley
Math. Department
Purdue University
West LaFayette, IN 47907

PROGRAM DEVELOPMENT METHODOLOGY

Sponsored by ACM/SIGCSE

A. J. Turner
Clemson University
Clemson, SC 29631

ABSTRACT:

Computer programs are often developed by users without considering the effort that will be required for someone other than the author(s) to modify it or to correct an error. Several techniques are available to assist in the development of computer programs that are easier to read, understand, debug, and modify. Since the techniques also facilitate the initial implementation of most programs, they are valuable even if a program is intended to be used without modification by its implementors.

Techniques for three aspects of program development are considered in this tutorial: design, programming, and implementation. Design techniques are used in the development of a program design, programming techniques are used in the development of the program code, and implementation techniques are used in the overall implementation process.

Topdown design is discussed as the basic approach to program design. Module independence, module function, information hiding, and the HIPO technique are considered as modularization criteria and paradigms to help develop the design.

Programming techniques that facilitate the development of program code are the use of pseudo code and stepwise refinement. Also included are techniques for improving the readability of program code, such as structured coding, and conventions for program format and comments.

The use of iterative enhancement and module stubs are discussed as implementation techniques that complement the design and programming techniques, and facilitate the use of a topdown approach to program development.

Emphasis is placed on the use of these techniques by a small implementation team or individual.

COLLEGE CAI PROJECTS IN THE BASIC SKILL AREAS

Susan M. Wood
George M. Bass, Jr.
Roger R. Ries
Helen Heller
S. A. Eveland
Michael G. Southwell

ABSTRACT: SIGI on Microcomputer

Susan Wood, Technical Director, SIGI, Educational Testing Service, Rosedale Road, Princeton, NJ 08541

The System of Interactive Guidance and Information (SIGI) was first designed and implemented at Educational Testing Service in the early 1970s by the Guidance Research team under the leadership of Martin R. Katz. A computer-based system, SIGI is designed to help students make rational and informed career decisions. SIGI provides a basis for acquiring knowledge and developing understanding. Through dialogue, simple modeling techniques, and exploration of different strategies or paths, the student can identify options, learn to interpret relevant data, and master decision-making strategies.

The project update will review the efforts to translate and distill a major software package consisting of over two megabytes of storage plus system program translations from Basic-Plus and Fortran into Pascal. We will have a demonstration using the TRS-80 Model II system for the turn-key SIGI guidance station at the conference.

ABSTRACT: Microcomputer-Assisted Study Skills (MASS) Project

George M. Bass, Jr. and Roger R. Ries, School of Education, College of William and Mary, Williamsburg, VA 23185

The major goal of this project is to create individualized instruction using present microprocessor technology to develop college students' study skills. Project MASS is directed toward accomplishing the following objectives:

1. To design, develop, and implement a microcomputer-based instructional system to assist students:
   - a. assess present study skills, attitudes, and habits;
   - b. organize and planning study time;
   - c. develop effective note-taking skills;
   - d. increase reading and paraphrasing ability;
   - e. develop effective test-taking strategies; and
   - f. improve library research procedures.
2. To assess the effectiveness of using microprocessor technology for individualized instruction with college students.

Our evaluation of Project MASS should provide formative data for developing instructional modules, and summative data on the effectiveness of these microcomputer instructional modules.

To accomplish this evaluation, participating students are observed, interviewed, and given criterion-referenced tests. In addition, the long-term effects will be measured through follow-up questionnaires and a review of these students' grade point averages.

ABSTRACT: The CLEF Project: Learning French on Colour Micros

Helen Heller, Faculty of Arts, The University of Western Ontario, London, Canada N6A 3K7

The CLEF (Computer-aided Learning Exercises for French) project began at The University of Western Ontario in 1978. A team of French instructors from the secondary and university systems set out to create a pedagogically sound program of introductory and intermediate French grammar drills to be used on microcomputers by students of all ages.

Our prime objective is writing two complete series of French CAL lessons--one for beginners, one for intermediate students--each of which contains drills on all the major grammatical points a student encounters at that level. Because we are convinced that a CAL program that covers the basics in a dynamic, exciting way can be used to advantage with any course of study, we have made our courseware text independent.

We seek to create quality pedagogical material uniquely suited to the medium of the micro screen. CLEF lessons are programmed in Basic and are intended for use on colour micros. Our programs use the microcomputers' flexible screen management, timed messages, simulated movement, graphics, cursor control, and colour to stimulate user motivation. Another major concern in coursewriting has been to provide immediate, detailed, constructive responses to the student.

ABSTRACT: SAIL, An Authoring Language

S. A. Eveland, Linguistic Research Center,
The University of Texas, P. O. Box 7249,
University Station, Austin, TX 78712

Using Sail to develop CAI modules in any subject is not a new concept; it provides more flexibility and is commonly available on DEC-10 and DEC-20 systems. Unlike Basic, it is a standardized language and is therefore readily transportable.

The software is a set of three programs: The first allows the instructor to build the module; the second program processes a cumulative record of all students' interactions produces a formatted file suitable for printing; the third program uses this cumulative record for some basic bookkeeping and then produces a report showing the high, low, and average grades.

The primary benefit of using these programs is the flexible formatting capability which allows questions to be unusually long. Additionally, the powerful file handling capabilities of Sail make detailed record keeping feasible. Furthermore, Sail is transportable and generally cheaper to run than languages like Basic.

ABSTRACT: The COMP-LAB CAI Modules

Michael G. Southwell, Department of English,
York College of the City University of
New York, Jamaica, NY 11451

At York College/CUNY a set of CAI modules have been developed to teach writing to nontraditional students (primarily minority students who have severe problems with correctness and clarity, usually because of nonstandard speech).

Computers are highly efficient in providing elements of language instruction: structured instruction and practice two. But many existing language-teaching computer programs have suffered from one or more of three faults: (1) providing structured instruction is terribly difficult and time-consuming, and so very little of it is available; (2) it's easy to provide non-contextual practice, though, and so lots is available (even though it has often been shown not to improve learning); and (3) practice (which should foster learning) and testing (which should check it) have often been confused.

The COMP-LAB CAI modules attempt to overcome some of these difficulties. The focus throughout has been on providing systematic instruction which encourages learning and on taking advantage of the computer's ability to manage that instruction. The amount provided is still too small to justify assessing its effectiveness. Although only one module (out of 12) is now available, and that only in a format suitable for CUNY's mainframe computer, the same lessons will be available in microcomputer formats before long. Development is continuing on other lessons.

COMPUTER ASSISTED
ACADEMIC DEPARTMENTAL
SCHEDULING

Ronald Prather
and
Ahcene Rabia
University of Denver

I. INTRODUCTION

Each year in every college or university department, someone is faced with the task of developing the next year's schedule of classes. Anyone on whose shoulders this task has fallen will admit how tedious a chore it can be, one to be avoided if at all possible. In an attempt to minimize the difficulty, several approaches to an automatic scheduling procedure have been suggested in recent years. With so many parameters to consider and with so many conflicting objectives at play, one can only hope for a computer approximation to satisfactory scheduling, one where human intervention would be encouraged in settling some of the more delicate issues at stake. Still, it can be argued that an automatic heuristic approach could be quite useful in arriving at a first approximation to one's final solution.

In more recent investigations, the overall development of schedules is best treated in two stages.[1] One first seeks to assign faculty members to courses or sections of courses according to a predetermined availability of faculty, a desirability of course offerings, and an array of preferences. In the second phase, one uses the above assignment, now identifying courses with faculty, to assign the courses to a collection of available timeslots. Note that we choose to disregard a related problem, that of assigning room numbers, because such a question is often outside of the scheduler's jurisdiction.

A network model has been announced and tested for handling the first phase (faculty to courses) of the scheduling.[2] The second phase seems to be more difficult, because of conflicts that may arise, e.g., a faculty member being required to be at two places at the same time. We would argue, however, that both phases can be handled quite effectively with a uniform appeal to the so-called Hungarian algorithm for the classical assignment problem. With a simple modification to the output routine, alternate assignments can be examined by the scheduler, eliminating any possible conflicts and imposing further value judgments of one form or another. Slight complications notwithstanding, the idea of using one main algorithm for both phases of the problem is a decided advantage from a computational standpoint. The simplicity of our computational model is a desirable feature, holding out the hope that our experience can be repeated successfully at other institutions.

II. THE CLASSICAL ASSIGNMENT PROBLEM

Our class scheduling task has obvious similarities to the classical assignment problem.[3] There we suppose that n persons are to be assigned to n jobs, and the array entry A_{ij} represents the "cost" of the i^{th} person in performing the j^{th} job. An assignment is given by a permutation:

$$a = \begin{pmatrix} 1 & 2 & \cdots & n \\ j_1 & j_2 & \cdots & j_n \end{pmatrix}$$

indicating that the i^{th} person is assigned to the job $j_i = a(i)$. An optimal assignment is one that minimizes the sum $\Sigma A_{ia(i)}$. We may think of the cost as a kind of inverse measure of effectiveness, i.e., the lower the cost, the better qualified is the individual for the job. In this way, one indeed seeks to minimize the above sum over all $n!$ possible assignments.

In a well known approach to the assignment problem, the following result, due to Konig and Egevary is used:

> In a rectangular 0, 1 array, the minimum number of "lines" (rows or columns) necessary to cover all of the zeros is equal to the maximum number of "independent" (no two on a line) zeros.[4]

One is led to a straightforward iterative technique known as the Hungarian algorithm, a method that is much preferred over the use of the linear programming techniques which would also be applicable.[5]

In order to have some appreciation for the type of computations involved in the Hungarian algorithm, a brief exerpt from an example is useful. In the midst of deriving an optimal assignment, one might be preparing to make a single step of the Hungarian method on the array:

$$\begin{array}{cc|cccc|l} & & \multicolumn{4}{c}{\text{jobs}} & \\ & & 1 & 2 & 3 & 4 & \\ \hline & 1 & 0 & 4 & 13 & 0 & \leftarrow L_1 \\ & 2 & 8 & 1 & 9 & 0 & \\ \text{persons} & 3 & 6 & 6 & 21 & 0 & \\ & 4 & 10 & 0 & 0 & 0 & \leftarrow L_2 \\ & & & & & \uparrow & \\ & & & & & L_3 & \end{array}$$

Here, three is the minimum number of lines (L_1, L_2, L_3) needed to cover all of the zeros. If this number is equal to n, the number of persons or jobs, an optimum assignment will have been reached. Otherwise, the minimal line covering is used as follows: *Choose the minimum array entry not covered by a line, to be subtracted from all such entries, and simultaneously, added to all entries at the intersection of lines.* In the present example, we would thus obtain the modified array:

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & \underline{0} & 4 & 13 & 1 \\ 2 & 7 & \underline{0} & 8 & 0 \\ 3 & 5 & 5 & 20 & \underline{0} \\ 4 & 10 & 0 & \underline{0} & 1 \end{array}$$

from which the (optimal) assignment:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 4 & 3 \end{pmatrix}$$

is determined.

The key to the method is the observation that the modified arrays have the same optimal solutions as does the original array, and that the number of independent zeros increases with each iteration. Nevertheless, the apparent simplicity in the calculation is something of an illusion, overlooking the difficulty of finding a minimum line covering (or equivalently, a maximum set of independent zeros) at each iteration. The most straightforward approach is found in the work of Yaspan, whose algorithm forms the computational cornerstone of our program for the scheduling problem.[6]

A recent article[7] shows that the most complete and realistic treatments of scheduling and timetable problems are exceedingly difficult to solve. In view of such pessimistic observations, and realizing that all previous investigations have quite specific and differing objectives in mind, we have tried to make a fresh start. Our idea is to attempt to capitalize on the underlying simplicity of the Hungarian algorithm and to exploit the similarity to the scheduling problem in the hope that the resulting solution will be sufficiently flexible for the intended applications.

III. The Scheduling Problem

In setting up a preferential assignment array for phase one of the problem (faculty to courses), a good idea is to weigh the faculty preferences against an administrative judgment. Thus, it may be that a certain faculty member strongly requests a given course, but the scheduler or some other administrative representative feels that such an assignment would not be in the best interests of the department. Accordingly, we can envision two sources of input to be weighted in generating the preference array:

$$P_{ij} = P^{(1)}_{ij} + P^{(2)}_{ij}$$

with

$P^{(1)}_{ij}$ = i^{th} faculty preference for j^{th} course

$P^{(2)}_{ij}$ = i^{th} faculty suitability for j^{th} course

The former is determined by a poll of the faculty, and the latter is generated from a confidential file of faculty effectiveness, perhaps based on student evaluations and other relevant data. In each case, we assume that the lower the rating, the greater the preference or suitability. We are indeed facing a minimization problem.

Supposing that a satisfactory solution has been determined for phase one, the second phase will consist of the assigning of (faculty, course) pairs to timeslots, again using a weighted preference array:

$$Q_{jk} = Q_{jk}^{(1)} + Q_{jk}^{(2)}$$

where

$Q_{jk}^{(1)} =$ j^{th} faculty preference for k^{th} timeslot

$Q_{jk}^{(2)} =$ j^{th} course suitability for k^{th} timeslot

These are derived by using further results of a comprehensive faculty poll and administrative information pertaining to the desirability of offering various courses at given times. The main advantage in treating the scheduling in two stages is now quite clear. We are able to inject faculty time preferences into the course-timeslot assignment problem through the identification of specific faculty members with courses.

We should point out that our arrays, P and Q, are in fact square arrays. In the case of P, we have a certain number, say n, of courses that we intend to offer. Here, we count multiple sections of a given course as if they were separate courses, i.e., as separate column headings. Similarly, a given faculty member will be duplicated as a row heading several times, depending on the number of courses or sections that he or she is expected to teach in the given period. Still, it is more than likely that the number of rows is less than n, and we simply fill out the array with staff until the array is square. We use entries of zero throughout the staff rows. Similarly, in the second phase of the assignment we will prescribe a number of timeslots (columns) somewhat in excess of the number of courses (now rows), and we add enough dummy courses as rows with zero entries to ensure that Q is also a square array.

It might be thought that student opinion has been left out of our equation, but that is not the case. The entries

$$P_{ij}^{(2)}$$

can be determined in part from past student evaluations of faculty effectiveness. Similarly, the coefficients

$$Q_{jk}^{(2)}$$

can be influenced by student preference. In a responsible determination of the second set of coefficients, the administration will be acting on the behalf of student interests, or at least, that is the intent.

We have already mentioned the primary reason for our treating the assignment in two stages. A further advantage is allowing for human intervention between phase one and phase two. One must be careful to guard against an over-reliance on the output of such a program. Experience shows that it is best to view such attempts at optimization as a heuristic aid, useful in the preparation of a schedule, but not as representing the last word on the subject. The questions are too delicate, and there are too many peculiarities to consider in any real department for us to believe otherwise. At some point the human scheduler must intervene. Having split the problem in two, the scheduler can make adjustments between the two stages before going ahead. Moreover, we will see in the next section that a provision is made for the ease of juggling alternatives at the end of either phase of the solution.

Excepting for such allowances, however, it is to be understood that the arrays P and Q are treated as assignment problems in the sense of Section II. In this way, we are able to bring a well-studied efficient algorithm to bear in handling both phases of the problem. We can kill two birds with one stone, so to speak. Any detailed discussion of the program is beyond the scope of this paper, particularly the treatment of the Yaspan procedure mentioned earlier. We suggest, however, that our program represents a novel use of the Hungarian technique in a most practical academic setting. Limited testing of the program in the Department of Mathematics and Computer Science at the University of Denver indicates that quite satisfactory assignments can be generated with the computational costs that are suprisingly modest.

IV. Input and Output Procedures

As we have indicated, a faculty poll must be taken in advance of the scheduling to determine individual preferences. Often the exact list of course offerings will not be known this far in advance, so it is better to over-estimate the offerings in drawing up the initial list of courses for the preferential poll. In any case, for each course we ordinarily ask the faculty to check one of the following responses:

would like very much	would like	neutral	don't really want	can't or won't
(1)	(2)	(4)	(6)	(9)

Underneath the responses are the coefficients that we use as the corresponding

$$P_{ij}^{(1)}$$

entries. The administrative ratings

$$P_{ij}^{(2)}$$

are given similar interpretations:

best	good	fair	poor	no!
(1)	(2)	(4)	(6)	(9)

These latter entries, rating all faculty for all of the courses in the departmenal catalog, are stored in a permanent confidential file. This file must be updated from year to year to reflect changes in interests, improvements in evaluation, etc.

After adding corresponding coefficients, we make a slight adjustment:

$$P_{ij} = P_{ij}^{(1)} + P_{ij}^{(2)} + \begin{cases} -2 & (\text{FP: } P_{ij}^{(1)} = 1) \\ -1 & (\text{AP: } P_{ij}^{(1)} = 1) \\ +1 & (\text{AP: } P_{ij}^{(1)} = 9) \\ +2 & (\text{FP: } P_{ij}^{(1)} = 9) \end{cases}$$

This is an attempt to use seniority as a tiebreaker. Thus, we subtract two when a full professor (FP) would like a certain course very much, but we only subtract one for associate professors (AP). An analogous modification is made at the other extreme when one of these faculty members can't or won't teach a given course. With this provision, the entries in our array P will vary from 0 to 20, a range that seems to be sufficient for making the necessary distinctions.

In yet another section of the faculty poll, we ask each instructor to indicate a general preference for teaching time periods, with one check in each row of the following table (to be modified, in case a department does not offer any evening courses):

	best	good	fair	poor	no!
early morning					
late morning					
early afternoon					
late afternoon					
evening					

To indicate severe restrictions, times that one would prefer not to teach, faculty will indicate by checks in the following table:

	M	T	W	Th	F
early morning					
late morning					
early afternoon					
late afternoon					
evening					

From this information, again interpreting information in the first table (best = 1, good = 2, etc.) and a check as a 9 in the second table, we are able to compute the faculty preferences

$$Q_{jk}^{(1)}$$

for individual timeslots. We use twenty-two types of timeslots, e.g.,

timeslot type 1 = 8:00 - 9:00 D
timeslot type 2 = 8:00 - 9:00 MWF
...
timeslot type 11 = 1:00 - 2:00 D
timeslot type 12 = 1:00 - 2:00 MWF
...
timeslot type 21 = 6:30 - 8:00 MW
timeslot type 22 = 6:30 - 8:00 TTh

As column headings for the Q array, the scheduler will choose a sufficient number of timeslots of the different types to be ensured of being able to handle all of the course sections now treated as row headings. More accurately, we should say that a row heading is a (faculty, course) pair. In this way, the faculty preferences can be accomodated as entries

$$Q_{jk}^{(1)}$$

after distributing the timeslot types into corresponding time periods:

	M	T	W	Th	F
early A.M.	1, 2, 3, 4,	1,3	1, 2, 3, 4,	1,3,	1, 2 3, 4
late A.M.	5,6,7, 8,9,10	5,7,9	5,6,7, 8,9,10	5,7,9	5,6,7 8,9,10
early P.M.	11, 12, 13, 14	11,13	11,12, 13,14	11,13	11,12 13,14
late P.M.	15, 19	16,20	15,19	16,20	
evening	17, 21	18,22	17,19	18,22	

For example, the j^{th} faculty member who has checked early afternoon as "fair" in the first table of the poll, and who has also checked the "evening T" entry of the second table, will have contributed

$$Q_{jk}^{(1)} = 4$$

to any timeslots of type 11, 12, 13, or 14 and

$$Q_{jk}^{(1)} = 9$$

to any timeslot of type 18 or 22.

Of course, this analysis treats only half of the story regarding the Q array. The administration must provide the other half, looking at a row heading as a course rather than as a faculty member. The details will vary from department to department, but there is always some rationale for preferring certain timeslots for certain courses. To the extent that such preferences can be quantified (as 1 = best; 2 = good; 4 = fair; 6 = poor; 9 = no!), as before, we are able to compute

$$Q_{jk}^{(2)}$$

for use in the expressions by using the seniority adjustments:

$$Q_{jk} = Q_{jk}^{(1)} + Q_{jk}^{(2)} \quad + \quad \begin{cases} -2 \text{ (FP: } Q_{jk}^{(1)} = 1) \\ -1 \text{ (AP: } Q_{jk}^{(1)} = 1) \\ +1 \text{ (AP: } Q_{jk}^{(1)} = 9) \\ +2 \text{ (FP: } Q_{jk}^{(1)} = 9) \end{cases}$$

As before, we obtain an array Q with entries in the range from 0 to 20.

In the use of either array, P or Q in the Hungarian algorithm, one is ordinarily led to the output of a single optimum solution, even if there are several assignments of equal minimum cost. And yet, we would like for the scheduler to be aware of all of the minor adjustments that could be made without changing the cost, or at least, with little change. For this reason, we arrange that our output routine actually displays a neighborhood of the optimum solution in a convenient and usable format.

In the final determination of an optimal assignment, we have a modified array, P or Q, where an independent set of n zeros identifies the solution in the form of a permutation. Speaking of the first phase of our problem, to say that the i^{th} faculty member is assigned to the j^{th} course is to say that $P_{ij} = 0$ in the modified array. Each column (course) will contain a zero, to be sure. But in general, there will be more than one zero in a given column and a number of near-zero entries as well. We arrange to print out all such information in the form:

```
320   DISCRETE STRUCTURES        PRATHER

     0     RECHARD

     2     MARTIN

     3     COHEN

     4     DORN
```

This indicates that the optimum solution assigns Prather to course 320, but that Rechard would do equally well, and that three other faculty could be assigned with little drop off in effectiveness. With this kind of information in hand for the complete list of courses, the scheduler can juggle the solution to see what changes can be made should something look out of line.

The same technique is used in the second phase of the problem, listing a neighborhood of timeslots around the optimum solution, e.g.,

```
320  DISCRETE STRUCTURES  1:00-2:00 D   PRATHER

   1    4:35 - 6:25  TTh

   2   11:00 - 12:00  D

   2    2:00 -  3:00  D

   3   10:00 - 11:00  D
```

Again, the scheduler can make any necessary adjustments. We note, however, that these timeslots are "keyed" to the choice of the instructor, making it somewhat difficult to see the effect of a change in faculty should this be necessary at the second phase. In fact, one hopes that very little of this will be required, otherwise it may be necessary to repeat the first phase again. This is one of the shortcomings of our method.

V. Discussion of Results

We are entering the second year using computer-assisted scheduling in the Department of Mathematics and Computer Science at the University of Denver. We might say that the program works so well that most of the faculty are unaware that they are being manipulated by a machine. On the other hand, from the scheduler's point of view, and speaking firsthand, it has transformed a tedious task into a rather enjoyable undertaking. But does it really work? Yes, we would have to say that one could have taken the unaltered optimal solution as a final copy, and it would not have looked all that unreasonable. Moreover, with the allowance for intervention on the part of the scheduler, quite satisfactory assignments are actually obtained. To be sure, one could argue that the inherent tedium of the chore has merely been replaced by the tedium of data preparation, the gathering of polling information, the processing of evaluations, the typing of files, etc. It is difficult to measure this trade-off effectively, but certainly the process is more orderly than before.

From an economical standpoint, one would like to be able to at least estimate the cost in computer processing time to be expected. The program is currently written in Algol for the Burroughs B6800. Our typical charge per run is as follows:

Phase I	3.6 sec.
Phase II	7.3 sec.
Total Charge	10.9 sec.

For twenty-four courses, this translates to a dollar figure of approximately $0.70. One must keep in mind, however, that the Hungarian procedure is an "order n^4" algorithm, so doubling the size of the department, i.e., the number of courses, one might expect a sixteen-fold increase in cost. Still, it would seem that even here there would be room for considerable experimentation while keeping cost within reasonable limits.

The program is quite flexible, and adaptable to departments of different size and with different requirements. In the interest of portability, we are currently rewriting the main portion of the algorithm in Pascal. Copies of the original program or the Pascal version are available from the authors upon request.

REFERENCES

1. Bloomfield, S.D. and McSharry, M.M., "Preferential Course Scheduling", Interfaces, Vol. 9, No. 4, 1979.

2. Dyer, J.S. and Mulvey, J.M., "An Integrated Optimization Information System for Academic Departmental Planning", Management Science, Vol. 22, No. 12, 1976.

3. Hall, M., Combinatorial Theory, Blaisdell Publishing Co., Waltham, Mass., 1967.

4. Ryser, H.J., Combinatorial Mathematics, MAA Carus Monograph Series, distributed by John Wiley & Sons, New York, 1963.

5. Kuhn, H.W., "The Hungarian Method for the Assignment Problem", Naval Research Logistics Quarterly, Vol. 2, No. 1 and No. 2, 1955.

6. Yaspan, A., "On Finding a Maximal Assignment", Operations Research, Vol. 14, No. 6, 1966.

7. Even, S. and Shamir, A., "On the Complexity of Timetable and Multicommodity Flow Problems", SIAM Jour. of Computing, Vol. 5, No. 4, 1976.

A FINANCIAL PLANNING MODEL

Dr. Robert L. Horton
Dr. Gary L. Eerkes
Gonzaga University

Introduction

The problem of declining resources and increasing expenditures has pervaded the entire structure of higher education. Deficit spending has abounded during the 1970s and the causes show no sign of abating. Demographic trends, continuing high inflation, cuts in federal and state funding, increasing costs for replacing and maintaining buildings and equipment, decreasing faculty turnover rates, and the decreasing ability of many families to afford higher tuition rates are guarantees that the financial crises will continue into the 80s. Meeting these challenges while continuing to provide quality instructional programs will require the careful management of an institution's financial resources.

Every size and type of institution has been affected by these problems, but the nation's small, private, liberal arts colleges and universities have been particularly vulnerable. It is estimated that as many as two hundred of these schools may be forced to close for financial reasons during the next decade.[1] In such an environment, financial decisions can no longer be made on a year-to-year basis. Detailed long-range forecasts must be prepared and all reasonable budget alternatives explored so that financial problems can be spotted early and informed decisions can be made to avoid them.

However, the task of preparing a budget and analyzing the various tradeoffs of tuition rate, faculty raises, capital expenditure, program quality, etc., is enormous, error-prone, and expensive if done by hand. But with the aid of a computerized model, this task can be done quickly, completely, and inexpensively. Using such a model, a financial planner can view the consequences of changes in specific income and expense items in a manner of minutes. The calculations are done automatically, eliminating the possibility of overlooking or mishandling important factors. Thus a wide range of policy options and contingency plans can be efficiently prepared, studied, and saved, taking a great deal of the uncertainty out of financial planning and forecasting. Usually a model doesn't actually prescribe any particular solution, but enables the planner to try out various courses of action before they are actually put into practice.

Alternatives

There are at least two nationally known programs of this type. The Resource Requirements and Prediction Model developed by the National Center for Higher Education Management Systems and the EDUCOM Financial Planning Model developed from a similar program used at Stanford.[2,3]

Yet despite the apparent need for such a facility and its availability, only 15-20% of the nation's higher education institutions use computer modeling as a financial planning tool.[4] Of course, the field is relatively new--the EFPM program has only been available since 1978. But for small schools in particular, these large-scale programs and networks may be inappropriate, too cumbersome, or too expensive. In such a situation, if the resources are available, a school must develop its own model, as was done at Gonzaga University during the summer of 1980.

Project Initialization

When Gonzaga installed a new HP 3000 with one megabyte of main memory in January 1980, the Board of Trustees recognized the absolute necessity of acquiring a financial modeling program to aid in both the annual budget preparation and in long-range forecasting. After investigating the available alternatives, Gonzaga decided that it would be more appropriate and cost effective to develop an in house program. Ownership, on-campus management, maintenence, and cost were major factors in the decision.

As outlined by the financial vice-president's office, the requirements for the program were quite specific:

1) The program must handle both the yearly budget planning and multi-year forecasting.
2) The program must follow the existing chart of account line items and subtotals and be compatable with existing budgetary reports.
3) The base year data must be stored in either a unit, unit rate, total form, or simply in a total form as appropriate.
4) The program must apply either percentage or absolute changes to base year data for up to a ten-year period with different changes for different years for the same line item.
5) The changes may be made to either individual line items or to subtotals. If to subtotals, these changes must be reallocated back to the appropriate line items on a percentage basis.
6) Various internal relationships specific to the institution must be handled.
7) The program must add or delete line items.
8) The program should be interactive.

A tacit assumption was that the program had to be simple to use and operate.

The authors wrote the Financial Planning Model in Basic during two months in the summer of 1980; it is now being tested. While the program itself is not easily transportable, relying on Gonzaga's specific chart of accounts structure, the approach to the problem and the design of the system is transportable and may be useful to those, particularly in the smaller universities, who are designing their own financial modeling tools.

Overview of the Model

Basically, the Financial Planning Model establishes a base year budget of revenue and expense line items and indicates the anticipated changes in any or all of the line items or groups of line items over a period of, at most, ten years. A budget forecast may then be calculated using these figures for the specified years. These forecasts may then be modified further or saved for future reference.

The model works with two types of files --scenario and assumption. A scenario file contains the actual data for the revenue and expense line items in either a unit, unit rate, total form, or simply in a total form for an eleven-year period (base year and ten succeeding years). An assumption file contains the anticipated changes in the line items over a ten-year period. These changes may be in either dollar or percentage form and can apply to either an single line item or to groups of line items. One file of each type may then be used to create a new scenario file that contains the base year values from the original scenario file and the projected figures based on the data in the assumption file. This process is called updating the scenario file.

These files may be manipulated in a number of ways. A new file of either type may be created or an existing file examined. Any file may be modified either by adding new items, or modifying or deleting existing items. Any assumption file may be used to update any scenario file. Any scenario file may be completed by calculating those line items which are dependent upon other budgetary line items and user-supplied figures. A new file may be saved at any time or an old file purged. A number of different reports involving individual line items or totals may be prepared for any file.

The program itself is menu driven and behaves like an operating system. There are eleven different digitized commands which the user can use singly. When the indicated operation has been performed, the program asks the user for the next command. By issuing the commands in the desired order, the user has complete control over the type and order of the operations performed on the files. These commands will be described in a later section.

File Structure

There are four major categories of files used by the Financial Planning Model. All but the last are HP KSAM files.

(1) Scenario Files - Each of these files contains the dollar figures for chart of account line items. Each record in the file contains a nine-character identification code, a thirty-character description, and values in either a unit, unit rate, total form, or simply in a total form for up to an eleven-year period. The chart of accounts is structured so that subtotals can easily be formed by examining subfields of the identification codes. These totals are automatically calculated by the system and stored in the scenario file in the same type of record as the individual line items.

(2) Assumption Files - These files contain the changes to be made in the corresponding scenario file records. Each record contains a nine-character identification code, a thirty-character description, and the modification values for up to a ten-year period. The changes may be represented either as percentages based upon the previous year or as the dollar figures to be inserted for that year. A code contained in each record differentiates between percentage and dollar figures.

(3) Program Workfile - This is a temporary file maintained by the system only for the duration of the session. As implied by the name, this is where the work on the files is done. An assumption or scenario file must be copied into this work area before any additions, deletions, modifications, or updates can take place.

(4) Catalog Library - This simple sequential file contains identification data for each of the permanently saved assumption and scenario files. Each record contains a file name, the file's description, and the dates of creation, last access, and last modification.

Commands

The Financial Planning Model is completely under user control. Any of the eleven possible commands can be issued whenever the command prompt appears. These commands include:

0 - DISPLAY

The command menu appears on the screen followed by the command prompt. This command is to be used whenever a command digit is forgotten.

1 - GET

This command brings any permanently saved, user-specified assumption or scenario file into the program's work area. A file must be in the work area before the commands MODIFY (3), UPDATE (4), COMPLETE (5), or READ (6) can be issued, since they pertain only to the work area data.

2 - WRITE

A new record will be written into the current file in the program's work area. The file may be either an assumption or a scenario file. If the work file is empty, this command is used to build a new file; if the work area already contains a file, the data must pertain to a nonexisting record. The user is polled for the appropriate data.

3 - MODIFY

An existing record in the program's work area may be modified or deleted. The user is again polled for the appropriate action, record identification code, and data.

4 - UPDATE

The actual forecasting is done using this command. When this command is issued, the program's work area must contain a scenario file. The user then selects an assumption file containing the changes desired from the catalog library. The assumption file is read record by record, and the indicated changes are made in the corresponding scenario record. Any nonexistent or incompatible records are listed on the screen, and a count of the total number of assumption records successfully processed is displayed.

5 - COMPLETE

There are several special records which depend upon other record values and upon budgetary figures and percentages supplied by the user. These are mainly benefit figures which depend upon various salary totals. This command enables those dependent values to be calculated and stored, thus completing the scenario file which is currently in the program's work area.

6 - READ

By issuing this command, records may be displayed on the screen from the current file in the program's work area without going through the more lengthy REPORT (7) procedure. The user is polled for the desired identification codes.

7 - REPORT

This command provides reports for any permanently saved assumption or scenario file, or of the catalog library file or of the work area file. The report content is completely under user control by selecting six parameter values. Anything from a single line item to groups of line items to totals to the entire file may be easily and efficiently obtained.

8 - SAVE

The file currently in the program's work area is saved in a permanent file location for later retrieval. The user is polled for the file name and description.

9 - PURGE

A user-specified assumption or scenario file is purged from the system.

999 - STOP

Program ends.

Design Strengths

Under this structure, the program is certainly simple to operate. A user need only be familiar with the eleven commands and have the necessary financial data available for input. But the design is still flexible enough to be a powerful planning tool. Any single set of base year data contained in a scenario file can be updated by any number of assumption files, thus allowing the financial planner to analyze several different courses of action by issuing a few simple commands. The forecasts may be only for the next year or may be for as many as ten succeeding years. Then these same assumption files can be used to update yet another scenario file containing an alternate allocation of income and expenses. Or a different set of assumption files can be chosen. In fact, any assumption file can be used to update any scenario file.

What would be the effect of a 10% faculty salary increase coupled with an 8% tuition increase and a 2% student enrollment decrease? What expenses will the new library generate ten years from now? What will the utility bill be in five years at the projected rates of inflation? These questions can all be answered in minutes, their impact on the total budget examined, and the answers stored for further study and additional modifications.

Or, a particular scenario file may be used to model only a subsection of the total university, such as a department. The same assumption files can still be used for updating or new assumption files can be created. The ten forecasted time intervals need not be years but could be days, weeks, months, decades, or whatever intervals seem appropriate.

Most important, the base-year data and the assumptions used to generate the forecasts are never lost. These files are available for study and use at any time until the user decides they are expendable, and specifically purges them.

Summary

Over the next decade, the well being and, indeed, the very survival of many institutions of higher learning will depend upon careful financial planning. Alternative long range allocations of resources must be examined in detail and yearly budgets planned accordingly, particularly at small, private, liberal arts colleges and universities. Computer modeling is necessary if this task is to be performed in an efficient, complete, and cost-effective manner. This paper presents the basic design of a financial planning program developed for a small university environment. It addresses both the annual budget preparation process and long-range forecasting. The structure is simple, yet flexible enough to provide an effective planning tool and may be of aid to those who have discarded the canned package or network approach and are designing their own financial models.

Bibliography

1. John Magarrell, The Chronicle of Higher Education, June 9, 1980.
2. Daniel Updegrove, "EFPM - The EDUCOM Financial Planning Model," EDUCOM Bulletin, Volume 13, Number 4, Winter, 1978.
3. J. B. Wyatt, J. C. Emery and C. P. Landis, Financial Planning Models: Concepts and Case Studies in Colleges and Universities, Princeton: EDUCOM, 1979.
4. Beverly T. Watkins, "How Stanford Uses Computer for Financial Forecasting," The Chronicle of Higher Education, March 17, 1980.

A USER-ORIENTED CLASSROOM USE PROGRAM FOR COLLEGES AND UNIVERSITIES

Dr. Larry Luce
North Texas State
University

There is a wide range of computer applications for the registration process. The more elaborate are complete systems which accomplish not only registration but also class scheduling. Such systems can be very efficient, but it is a big step socially and economically to implement an active registration system. In a more conservative mode there are systems which are passive by nature. They merely report the status of affairs and leave to individuals responsibility of making each decision and change.

There are several advantages of the passive systems:

1) They do not force changes in the way things are done. Thus they do not pose a severe threat to the individuals concerned.
2) Their use can be incorporated gradually into existing procedures. The old system serves as a back up during the transition.
3) Computer problems don't shut down the operation.
4) They are less expensive.

This paper discusses a passive room-use reporting system developed at North Texas State University. It is not a state of the art system, but does provide information for those who are interested in bringing much greater efficiency to a manual system.

At NTSU classrooms are not "owned" by the department, but are in one institutional pool. However, departments accustomed to using certain classrooms are generally allocated exclusive use and variable partial use of others by a central scheduling office. In effect, the central office accepts the schedules of the departments unless a conflict or problem develops.

The following criteria were set as goals of the system:

1) The output should be easily understood.
2) The program must be capable of being run at any time during registration and thereafter.
3) The programs must provide direct assistance to individuals seeking the most suitable available space for group instruction.

Since the program was designed for the user, we started with output formats and literally designed printout forms by hand. The forms were modeled to provide answers to user questions:

A. From the faculty—
    1) My class is all the way across campus. Do you have a (appropriate size and quality) room available in my building?
    2) OK, so my building is full at that hour! How about the building next door?
    3) OK, so it's full too! What is available?
    4) There are more students in my class than the room can hold. What larger rooms are available?
B. From the department chairmen and deans—
    1) We could enlarge our sections if two large rooms are available somewhere on campus. Are they available (or could small classes be moved out of them)?
    2) I need more offices. Can we teach some classes elsewhere and convert some of our classrooms to offices?
C. From the administration—
    1) How effectively are we using classroom and lab space?

2) What changes, if any, need to be made?

In most cases, the output formats combine a graph and numerical data. There are seven basic formats, although a particular one may be segmented into classrooms, laboratories, and a combination of the two.

1) Facilities never scheduled
2) Use by building, room, and half-hour time segment
3) Use by capacity and half-hour time segment
4) Use by building, room, and how well the room is filled
5) Use by how well the room is filled
6) Number of rooms in use per half-hour time segment
7) Available times and time combinations of rooms

The half-hour time segment is used because it allows tracking of either hour or hour and a half classes. All of the examples shown in this paper are classrooms. Similar runs can be made for class laboratories.

Exhibit 1, Classrooms Never Used, lists classrooms which were never formally scheduled for use. An individual review of the rooms may show that it was removed from assignment to accommodate a conference, an administrative training series, or some other valid use. A review might also show that the room no longer has classroom furniture at all, but has been quietly converted to storage, private conference room, graduate student office, etc., without authorization.

Exhibit 2, Classroom Use by Building, Room, and 30 Minutes of Use, shows all classrooms in each building, the capacities, and each 30 minutes of scheduled use represented by an X. A glance reveals how heavily a room or a building is used, and a closer look quickly shows what rooms are available at a particular time. Questions such as "What's available in my building?" can be answered with this report.

Exhibit 3, Classroom Use by Capacity and 30 Minutes of Use, ranks the classrooms by capacity. This allows the availability of all rooms of a given capacity to be examined easily.

Exhibit 4, Classroom Use by Building, Room, and Percentage Filled, shows how well the class size matches the room size. Listed by building, this information not only is information on how well we are doing, but also is a tool for trading rooms to achieve a better fit of classes to rooms.

Exhibit 5 takes the same data as Exhibit 3 and ranks it in ascending percentage filled. It is used not only as a measure of efficiency, but as a tool for rearrangement of classes. Both of these reports contain complete quantitative data on the class and room sufficient to make rearrangements.

Exhibit 6, The Percentage of Classrooms Per Building Used, displays the intensity of room scheduling throughout each day of the week. A glance may indicate if rooms are available early in the morning, at noon, or late afternoon. It not only serves as a general measure of room use, but can be effectively used as objective data in discussing a department or college's need for additional space, an ever present topic.

Exhibit 7, Room Utilization by Type and Capacity, is one of only two formats not using a graph. It reflects the use of classrooms of various sizes. Combined with a little caution and cross reference with Exhibits 3 and 4, it is a good evaluator of the array of adequate classrooms available.

Exhibit 8, Rooms Available by Time, Building, and Capacity, takes the first step and puts together the various common time sequences likely to be needed and displays them by building and capacity. Since few demands for time will be only once a week, the combination may be a useful automatic first step. It contains the same data as the other reports, but arranged in a different way.

None of these reports, singly or in unison, can replace a personal knowledge of the facilities on a campus or the needs of the people who use the facilities. They are only a valuable asset in the process of routinely and quickly achieving a better match of rooms and groups.

The reports can be produced separately, together, or in any combination. As previously noted, individuals within a given institution tend to concentrate on different reports as a result of their interests and responsibilities.

Exhibit 1

N.T.S.U. FACILITIES PLANNING & UTILIZATION

CLASSROOMS NEVER USED

	ROOM #	TYPE	CAPACITY	AREA	USAGE	%	DEPT
BUSINESS BUILDING	178	110	50	832	10	100	CLRM
MATTHEWS HALL	255	110	10	307	10	100	CLRM
GENERAL ACADEMIC	202	110	10	160	10	100	CLRM
	301	110	16	353	10	100	CLRM
	312	110	16	266	10	100	CLRM
JOURNALISM BUILDING	206	110	18	674	10	100	CLRM
INFORMATION SCIENCES BLDG	206	110	12	391	10	100	CLRM
WOMEN'S GYMNASIUM	125	110	32	519	10	100	CLRM
LANGUAGE BUILDING	301	110	36	564	10	100	CLRM
WOOTEN HALL	261	110	12	226	10	100	CLRM
HIGHLAND HALL	109	110	30	442	10	100	CLRM

Exhibit 2

N.T.S.U. FACILITIES PLANNING & UTILIZATION

```
CLASSROOM USE BY BUILDING, ROOM, & 30 MINUTES OF USE                 FROM 08:00 AM TO 05:00 PM

|BLDG   ROOM   CAPA-|      MONDAY      |     TUESDAY      |    WEDNESDAY     |     THURSDAY     |
|ABBR.   NO.   CITY |8         1       |8         1       |8         1       |8         1       |
|                   |------------------------------------------------------------------------|
|AUDB    201      28|xxxxxxxxxxxx  xx  |  xxxxxxxxx       |xxxxxxxxxxxx  xx  |   xxxxxxxxx      |
|        202      30|  xxxxxxxx  xx    |  xxxxxxxxx       |  xxxxxxxx  xx    |   xxxxxxxxx      |
|        207      28|  xxxxxxxx  xx    |  xxxxxxxxx       |  xxxxxxxx  xx    |   xxxxxxxxx      |
|        212      32|  xxxxxx          |  xxx             |  xxxxxx          |   xxx            |
|        217      32|xxxxxxxxxxxxxx    |xxxxxxxxx         |xxxxxxxxxxxxxx    |xxxxxxxxx         |
|                   |                  |                  |                  |                  |
|BUSI    116     186|xxxxxxxxxx        |xx xxxxxxxxx      |xxxxxxxxxx        |xx xxxxxxxxx      |
|        166      52|  xxxxxxx xxx     |xxxxxxxxx xxx     |  xxxxxxx xxx xxxx|xxxxxxxxx xxx     |
|        175      20|            xxxxxx|            xxxxxx|            xxxxxx|xxx               |
|        176      80|xxxxxxxxxxxxxxx   |xxxxxxxxxxxx      |xxxxxxxxxxxxxxx   |xxxxxxxxxxxx      |
|        178      50|                  |                  |                  |                  |
|                   |                  |                  |                  |                  |
|CHEM    121     125|  xxxxxxxxxx      |   xxxxxxxx xx    |  xxxxxxxxxx      |   xxxxxxxx       |
|        207      63|xxxxxxxxxx      xx|   xxxxxxxxxxx  xx|  xxxxxxxxxxxx    |   xxxxxxxxxxx  xx|
|        319      54|xxxxxx  xx        |   xxxxxx   xx  xx|xxxxxx  xx        |   xxxxxx xxxx  xx|
|                   |                  |                  |                  |                  |
---------------------------------------------------------------------------------------------------
```

Exhibit 3

```
                                      N.T.S.U. FACILITIES PLANNING & UTILIZATION
CLASSROOM USE BY CAPACITY AND                FOR MONDAY THRU FRIDAY              FROM 08:00 AM TO 05:00 PM
  30 MINUTES OF USE

  |BLDG   ROOM   CAPA- |      MONDAY      |     TUESDAY      |    WEDNESDAY     |     THURSDAY     |
  |ABBR.  NO.    CITY  |8         1       |8         1       |8         1       |8         1       |
  |                    |------------------|------------------|------------------|------------------|
  |WH     211      30  |  xxxxxx          |   xxxxxx         |  xxxxxx          |   xxxxxx         |
  |WH     213      30  |xxxxxxxxxx        |   xxx            |xxxxxxxxxx        |   xxx            |
  |AUDB   212      32  |  xxxxxx          |   xxx            |  xxxxxx          |   xxx            |
  |AUDB   217      32  |xxxxxxxxxxxxxx    |xxxxxxxxx         |xxxxxxxxxxxxxx    |xxxxxxxxx         |
  |BUSI   233      32  |  xxxxxxx         |   xxxxxx   xxxxxx|  xxxxxxx         |   xxxxxx         |
  |MATT   320      34  |xxxx    xx        |   xxxxxx         |  xxxx    xx xxxxx|   xxxxxx         |
  |GAB    204      34  |xxxxxx  xx        |xx xxx   xxx      |xxxxxx  xx        |xx xxx   xxx      |
  |BUSI   333      35  |xxxxxxxxxxxx      |xxxxxxxxxxxxxxxxxx|xxxxxxxxxxxxxxxxxx|xxxxxxxxxxxx      |
  |MATT   216      35  |xxxxxx    xxxx xxx|xxxxxxxxxxxxxxxxxx|xxxxxx    xxxx xxx|xxxxxxxxxxxxxxxxxx|
  |MUSI   287      35  |xxxxxxxxxxxx    xx|xx xxxxxxxxxxxx xx|xxxxxxxxxxxx    xx|xx xxxxxxxxxxxx xx|
  |MATT   316      36  |            xx    |      xxxxxx      |            xx  xx|      xxxxxx     x|
  |S/D    273      36  |  xxxxxx  xxxx    |   xxxxxxxxxxxx   |  xxxxxx  xxxx    |   xxxxxxxxxxxx   |
  |S/D    274      36  |    xxxxxxxxxx    |xxxxxxxxx   xxxxx |    xxxxxxxxxx    |xxxxxxxxx   xxx   |
  |                    |                  |                  |                  |                  |
  ----------------------------------------------------------------------------------------------------
```

Exhibit 4

```
                                          N.T.S.U. FACILITIES PLANNING & UTILIZATION
CLASSROOM USE BY BUILDING, ROOM,                   FOR MONDAY THRU FRIDAY              FROM 08:00 AM TO 05:00 PM
  AND PERCENTAGE FILLED

BLDG  ROOM     ---ACADEMIC--- ----CLASS----       STDNT  CAPA-  %        1    2    3    4    5    6    7    8    9   10
ABBR   NO.     DEPT CRSE SCTN DAYS     TIME       QNTY   CITY  USE       0    0    0    0    0    0    0    0    0    0
                                                                   --------------------------------------------------
BUSI   354     ACIS 361   009 MWF  10:00-10:50     38     46   083   ******************************************
       354     ACIS 361   010 TR    8:00- 9:20     40     46   087   ********************************************
       354     ACIS 361   011 TR    9:30-10:50     38     46   083   ******************************************
       354     ACIS 361   013 TR   12:30- 1:50     37     46   080   ****************************************
       354     ACIS 462   001 TR   11:00-12:20     25     46   054   ***************************
       354     ACIS 462   002 T     2:00- 4:50     33     46   072   ************************************
       355     FINA 377   004 MWF  11:00-11:50     66     65   102   ***************************************************

CHEM   121     BIOC 200   001 W    12:00-12:50     19    125   015   ********
       121     BIOC 454 * 001 TR   11:00-12:20     85    125   068   **********************************
       121     BIOC 456   001 M    12:00-12:50     35    125   028   **************
       121     CHEM 131   001 MWF   9:00- 9:50     61    125   049   *************************

NOTE:  AN  *  BY COURSE INDICATES TWO OR MORE CLASSES WERE COMBINED DUE TO DUPLICATE MEETING TIME AND PLACE
```

Exhibit 5

```
                                N.T.S.U. FACILITIES PLANNING & UTILIZATION
CLASSROOM USE IN ORDER OF                 FOR MONDAY THRU FRIDAY           FROM 08:00 AM TO 05:00 PM
  PERCENTAGE FILLED

BLDG  ROOM    ---ACADEMIC--- ----CLASS----        STDNT  CAPA-  %       1    2    3    4    5    6    7    8    9   10
ABBR   NO.    DEPT CRSE SCTN DAYS      TIME        QNTY  CITY  USE      0    0    0    0    0    0    0    0    0    0
                                                                   --------------------------------------------------
LANG   104    GERM 433   001 MWF   12:00-12:50       2     36  003 *
ART    219    ART  251   001 MWF   11:00-11:50       5     45  011 ******
BUSI   166    MGMT 682   001 W      3:00- 6:00       6     52  012 ******
BUSI   235    ACIS 543   002 MW    12:30- 1:50      16     48  033 ****************
CHEM   319    CHEM 545   001 TR    11:00-12:20      21     54  010 *********************
ISB    203    LIBR 622   001 W      9:30-12:20      20     40  050 **************************
PHYS   104    PHYS 342   001 TR    12:30- 1:50      39     71  055 ****************************
BIOL   200    BIOL 420   001 TR    12:30- 1:20      63    105  060 ******************************
MATT   319    EDSS 665A  001 T     12:30- 3:20      35     56  061 *******************************
BIOL   101    BIOL 501   501 TR     2:00- 4:50      71    105  068 *********************************
BIOL   200    BIOL 413   001 MWF    9:00- 9:50      75    105  011 ***********************************
LANG   104    FREN 505   001 T      2:00- 4:50      29     36  080 *****************************************
LANG   302    SPAN 565   001 MW     3:30- 4:50      35     36  097 ************************************************
BUSI   116    BCOM 333   202 F      8:00- 8:50     191    186  102 **************************************************
BUSI   328    MSCI 519   001 T      2:00- 5:00      51     54  110 **************************************************
```

Administration

65

Exhibit 6

N.T.S.U. FACILITIES PLANNING & UTILIZATION

PERCENTAGE OF CLASSROOMS PER BUILDING USED FOR MONDAY THRU FRIDAY FROM 08:00 A.M. TO 05:00 P.M.

BUSINESS BUILDING 24 ROOMS AVAILABLE

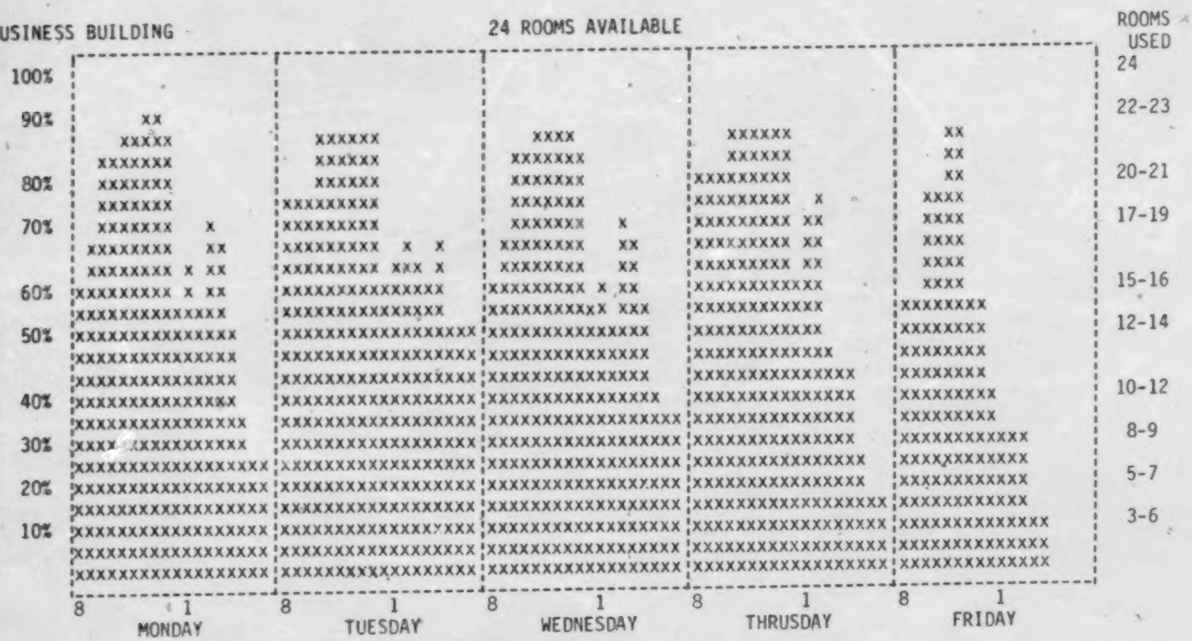


Exhibit 7

N.T.S.U. FACILITIES PLANNING & UTILIZATION

ROOM UTILIZATION BY TYPE & CAPACITY RANGE FOR MONDAY THRU FRIDAY FROM 08:00 AM TO 05:00 PM

ROOM DESCRIPTION	# OF ROOMS	CAPACITY RANGE	AVERAGE CAPACITY	--HOURS/WEEK---		PERCENT USAGE	AVERAGE STATION OCCUPANCY	AVERAGE STATION UTILIZATION	AREA PER STUDENT STATION
				AVAILABLE	USED				
CLASSROOM	21	11-20	15.7	945.0	223.5	23.6	87	21	21.7
CLASSROOM	43	21-30	28.7	1935.0	784.5	40.5	74	30	19.8
CLASSROOM	76	31-40	36.1	3420.0	1635.5	47.8	70	34	15.6
CLASSROOM	41	41-50	45.7	1845.0	1031.0	55.8	74	41	16.9
CLASSROOM	22	51-75	60.0	990.0	499.5	50.4	55	28	15.5
CLASSROOM	4	76-100	87.8	180.0	97.5	54.1	50	27	20.4
CLASSROOM	10	101-150	120.0	450.0	229.0	50.8	52	26	11.9
CLASSROOM	4	151-999	209.3	180.0	95.5	53.0	40	21	11.5

Exhibit 8

```
                        N.T.S.U. FACILITIES PLANNING & UTILIZATION

AVAILABLE LABORATORIES BY TIME, BLDG.,       FOR MONDAY THRU FRIDAY         FROM 08:00 AM TO 05:00 PM
  AND ROOM CAPACITY

   ********************************** EMPTY LABORATORIES 08:00 AM ****************************************

        DAYS             CAP    ROOM#        CAP   ROOM#         CAP   ROOM#         CAP   ROOM#         CAP   ROOM#
   0153 ART BUILDING
        MTWRF   MTWRF     5      109A  MTWRF   7    109C  MTWRF   12    111   MTWRF   18    221   MTWRF   18    240
        MWF       MWF    15      321     MWF  22    331     MWF   24    234     MWF   24    322
        TR         TR    20      231      TR  20    242      TR   24    313
        OTHER     TRF    22      115

   ********************************** EMPTY LABORATORIES 08:30 AM ****************************************

        DAYS             CAP    ROOM#        CAP   ROOM#         CAP   ROOM#         CAP   ROOM#         CAP   ROOM#
   0104 BUSINESS BUILDING
        MTWRF   MTWRF     7      151   MTWRF  40    153   MTWRF   40    334   MTWRF   48    152   MTWRF   75    154
        MWF       MWF    40      336
        TR         TR    30      330      TR  40    335
   0105 CHEMISTRY BUILDING
        MTWRF   MTWRF    19      112   MTWRF  20    220   MTWRF   20    312   MTWRF   25    309   MTWRF   28    215
        OTHER    MTRF    28      212
```

DEVELOPING COMPUTER LITERACY AT A LIBERAL ARTS COLLEGE

by

Nancy H. Kolodny and Gene Ott
Wellesley College, Wellesley, Mass. 02181

The opinion that computing should play some role in an undergraduate liberal arts education is now generally accepted. Many educators are now trying to decide what that role should be and how to effect the changes necessary to provide for computing at their institutions. Five years ago Wellesley College addressed these issues by establishing a computer-literacy project. The project has been monitored by an extensive evaluation program that shows dramatic changes in the nature of computer use and the attitudes of students, faculty and administration toward computing at Wellesley. In the hope that others may benefit from our experiences, we have prepared complete reports on Wellesley's project, which are available upon request. This paper summarizes our project and its results, with emphasis on the choices we made and why we made them. To be useful, this information must be interpreted in relation to the conditions prevalent at the local institution.

Wellesley College is a four-year liberal arts college dedicated to providing high quality education for women. The academic community consists of approximately 2200 students and 280 faculty members. Courses are offered in three major discipline groups: the humanities, the social sciences and the natural sciences. Wellesley does not offer any graduate studies or undergraduate courses in engineering or technology. It is in this context that the role of computing was established to be an enhancement of the existing curriculum rather than an independent activity.

In 1975, Wellesley College committed itself to a Computer-Literacy Project with the goal of creating and sustaining an academic environment in which faculty members and students are familiar with computers, their capabilities and limitations, and have specific knowledge and experience using computers in their particular disciplines. We intended to achieve this objective by integrating the use of computers throughout the existing curriculum so that students would learn about computers within the context of meaningful applications.

After investigating the problem of developing a computer-literate environment at other educational institutions and analyzing the situation at Wellesley College, we determined that to achieve our goal would require a combination of:

a) adequate computer facilities (including user services) available to academic users on a "free access" basis;

b) a computer-literate faculty interested in using computers in their courses; and

c) relevant and effective courseware (computer-based curriculum materials).

Consequently, our plan called for activities broadly classified as establishing and maintaining adequate computer facilities, providing user services, and conducting an educational program for students and members of the faculty. To help implement this plan we applied for and received a CAUSE grant from the National Science Foundation in 1976.

At that time active faculty interest in computing at Wellesley College was represented by the ad hoc Academic Computer Users' Committee, which consisted of 12 junior faculty members and 3 students. Most faculty members were either indifferent to computing or had only a latent interest. Our CAUSE grant gave the goal of computer literacy great visibility, and faculty reaction was very divided. Objections to the project fell into the following two main categories:

a) The cost of providing the resources necessary to develop and sustain a computer-literate environment at a time of difficult financial circumstances would consume funds that would otherwise be available for other needs of the College.

b) To some, computers at Wellesley represented an encroachment of technology that was not appropriate for a liberal arts college and might be used in educationally undesirable ways.

During the 1976-77 academic year there was considerable discussion of these issues among members of the faculty and administration. The project directors held several meetings to describe the project's benefits to Wellesley and to assuage faculty concerns. Support for computer literacy has grown steadily since then, as more faculty members have realized the benefit of computing in all disciplines of the College, with the most dramatic changes in attitudes occurring among faculty members in the humanities departments.

Prior to receiving our CAUSE grant, the academic computing facilities at Wellesley College consisted of six rented terminals located in a room in the basement of the administration building. These terminals were connected to the Dartmouth Time Sharing System (DTSS) via the NERComP (New England Regional Computing Program) network. We have since implemented our proposed plan to develop computing facilities appropriate and adequate for our academic needs. This plan included a computer on campus and connections to external computers via networks such as NERComP.

After a thorough study and evaluation of the alternatives, we chose a DECsystem-20 with the TOPS-20 operating system to serve our academic needs. In August 1977 "DECstar" (DECsystem-20 for teaching and research) was installed in a newly constructed computer room in the Science Center. Its "personality" was established to be polite and friendly, not obscure, condescending or cute.

New construction included two public terminal rooms, one in the Science Center adjacent to the computer room and one 1/4 mile away in the Clapp Library (the main college library). The equipment in each public cluster now includes at least ten terminals and a line printer. There are graphics terminals and a flatbed plotter in the Science Center facility. Other terminals are located in individual academic departments and laboratory areas. We now have more than 40 terminals on campus and several faculty members have terminals at home.

When we received the CAUSE grant, the Computer Science Department consisted of the director (faculty), a full-time assistant (non-faculty), and 12 student consultants who could each work five hours per week. To maintain the new local computing facilities and provide the user services required for the project, the position of assistant to the director was replaced by two positions, a software analyst to maintain DECstar and a user services coordinator to manage a team of 26 student consultants and help users with special problems. The student consultants help users at terminals in our two public clusters. Faculty members in the department teach the computer science courses, but are not expected to provide specific services to users. However, as part of a grant from the Alfred P. Sloan Foundation, we have a courseware specialist on the staff who helps faculty members develop computer-based curriculum materials.

The spectrum of services and technical support provided by student consultants and members of the Computer Science Department includes:

a) maintaining sources of information for users, including manuals and HELP files;

b) assisting users in our two public terminal clusters;

c) consulting on statistical and programming problems, particularly for research projects;

d) consulting on computer hardware, software and communications for persons with individual dedicated systems;

e) teaching classes for individual instructors, weekly sessions for new users, and mini-courses on topics of general interest;

f) helping to locate appropriate software, including datasets; importing software and performing conversions;

g) writing small applications programs which require more expertise than the user can be expected to develop in the time available;

h) developing new software for general use and for specific curricular purposes; and

i) performing all tasks necessary to maintain the DECstar facilities, including daily backup of disk storage.

With these new computer facilities and services available, we expected computer use in the curriculum to increase due to:

a) spontaneous use by knowledgeable faculty members with little or no help from the professional staff;

b) use by interested faculty members who have their own ideas, but need technical assistance;

c) spread of use within a department as a result of the efforts of one or more members of the department; and

d) missionary work by the staff of the Computer Science Department to educate faculty members to the possibilities of computer-based instruction.

However, many faculty members were inhibited by their lack of confidence in their own knowledge about computing as well as how to use it effectively in their courses. This concern was particularly true of the senior faculty members who provide the greatest stability to the educational process. To solve this problem we developed a plan to provide an appropriate opportunity for faculty members to become computer literate and an effective mechanism to document and implement their courseware needs. This three year "Faculty Participation Program" was funded by the Alfred P. Sloan Foundation in February 1978.

Computers are now used throughout the curriculum at Wellesley as:

a) a teacher (computer-assisted instruction and computer-managed instruction);

b) a student (programming);

c) an environment (simulation);

d) a computational tool (information processing); and

e) a laboratory tool (experiment control and data collection).

A selection of specific examples is presented below.

a) Although CAI materials would be very useful for some purposes, we have not encouraged their use for several reasons. Some of these reasons are intrinsic to the nature of CAI (as we use the term here) and some are peculiar to an environment like Wellesley. In particular, as a small college with a relatively low student-teacher ratio, we do not seek to replace the teaching function of the faculty or alter the close student-teacher relationship that underlies the basic pedagogical philosophy of the College.

At present we have an English grammar package and a writing package that are used for remedial purposes. Some of our foreign language teachers have implemented CAI materials for their beginning students, using Common Pilot as the authoring language. Our CRT terminals have been modified to include the overstrike characters necessary to display the Romance languages, German, and Romanized Chinese.

b) An increasing number of students each year are taking our three programming courses. In these courses they learn to program in Basic, Fortran, and Pascal. In addition, students are being taught programming and are required to write programs in many other courses throughout the science curriculum. Students who learn programming early in their college careers tend to write programs to solve problems in other courses later on. Students also write programs for extracurricular projects, such as the production of a "Student Course Guide."

c) The computer is used as an environment via simulation programs. Such simulations are useful when real experiments are too expensive, time consuming, or dangerous. They allow a faculty member to separate data collection from experiment design and data analysis to more efficiently use students' time and to give more challenging assignments.

An unusual way of using DECstar as an environment is provided by our program SEMINAR. SEMINAR allows the "discussion" of issues with DECstar as the mediator. A faculty member enters a series of questions or statements on a given topic and gives students the information they need to participate. Students participate at their own convenience by typing their responses to the questions or statements into the computer. They may be shown other students' responses, asked to rate them, and be allowed to change their own responses. The SEMINAR may be anonymous or not, as the faculty member chooses. At the end of the SEMINAR, the faculty member obtains a written copy of the entire "discussion" and summary statistics which are brought into the classroom to stimulate further discussion.

d) The most common use of DECstar is as a computational tool or information processor. Our extensive library of programs allows faculty members to make assignments without having to create new software. Most students are introduced to the computer via existing programs which are highly interactive and easy to use. These programs range from universal statistical tests, used extensively by students in psychology, sociology, and economics, to specialized statistical analyses for geology or biology, to manipulations of data bases in chemistry for the identification of unknown organic compounds based on their chemical and physical properties. Sophisticated statistical packages, such as SPSS, are also used in research courses and independent projects.

Using a powerful editor called EMACS, students and faculty are writing and editing their papers and theses on the computer. Students find it easier to write papers with EMACS and report better grades for their efforts. The growing popularity of this word processing application of DECstar has been quite spontaneous.

e) Our PDP-11/34 in the Psychology Department is used as a laboratory tool for experiment control and data collection. In addition, students are using microcomputers in research projects with faculty members in the Biological Sciences and the Chemistry Departments.

The time and effort that faculty members have given to learn to use our computer facilities and integrate computing into their courses, combined with student interest and independent initiative, have lead to a substantial growth in computer use. Several indicators which illustrate this trend are tabulated below.

To quantify the progress we have made towards a computer-literate environment at Wellesley College, we have used

	Academic Year					
Computer Users	75-76	76-77	77-78	78-79	79-80	
Students	650	800	818	1,126	1,318	(2,283)
Faculty	50	30	45	86	124	(288)
Courses	42	29	73	119	239	(575)
Departments	13	10	13	17	23	(28)
Connect hours	5,200	5,800	9,600	17,300	33,000	

Notes: Numbers in parentheses are the 1979-80 totals in each category. Many courses at Wellesley are taught in both semesters and in multiple sections per semester. The numbers given count all sections of a course as one course.

the following index of literacy levels. These levels are not intended to reflect our concept of what constitutes computer literacy, but do represent a measurable indicator of user sophistication.

Level 0 -- No use of a computer.

Level 1 -- Know how to sign on and use an interactive program.

Level 2A -- Know how to program in at least one language available at Wellesley. Have written a program to solve a problem.

Level 2B -- Know how to use a sophisticated software package such as SPSS, SCSS or STATPACK. Have used the package for an "independent" project.

Level 3 -- Know how to program well. Have written a program to be used by others.

The computer, which maintains a record of log-ons, automatically assigns each student a literacy level of 1 or 0. Assignment to the other levels is done on the basis of the courses in which students are enrolled. A questionnaire completed by faculty members is used to assign a literacy level to the computer use in each course. Class lists obtained from the Registrar's Office provide the names of all students in these courses.

The pie charts of Figure 1 present the results of using this procedure to assign a computer literacy level to each of the students in the Class of 1979. The percentage of students who had used the computer in any way (level 1 and above) is 90% for science majors, 87% for social science majors, and 66% for humanities majors. The percentage of all students at level 1 or above is 80%. Our goal of having 90% of graduating students qualified at level 1 or higher was achieved for the scientists, and almost achieved for the social scientists. We are still striving to achieve this goal for humanities majors.

A literacy level of 2 or above was attained by 62% of natural science majors. Among the natural science departments, chemistry had the highest percentage of majors at level 2 or above (81%), while biological ciences had the lowest (54%). Also, biological sciences majors tended to attain level 2B, rather than 2A, probably because they were more likely to be involved in research projects that required using sophisticated statistical packages rather than in projects which required programming.

Of the social science majors, 68% achieved level 2 or higher. Political science and ociology majors attained level 2 at the rate of approximately 60%. In economics and psychology, over 90% of the majors qualified at level 2 or above, largely as the result of courses required for the major in which students use sophisticated statistical packages. It is interesting that about 25% of economics majors and 20% of psychology majors at level 2 both wrote computer programs and used sophisticated statistical packages. This combination will probably be more prevalent in the future, as more social science majors decide to take courses in programming.

Our 1980 faculty workshop was attended by members of the anthropology, philosophy, and religion departments as well as the humanities departments. We expect our results in 1980-81 to show an increase in sophistication and amount of computer use in their departments as a result of the efforts of these faculty members.

To make our experience more useful to others, we have tried to identify the significant factors which contributed to the success of the project. Those which preceded the implementation of our project include:

a) articulation of the value of computer literacy by President Barbara Newell and a small but active group of faculty members and students;

b) identifying specific individuals as the project directors to be responsible for the development of computer literacy, with the advice of the Academic Computer Users' Committee; and

c) formulating and documenting the project goals and a plan to achieve them, including a thorough cost analysis.

Although all elements of the plan were important, the most critical were a commitment to provide adequate, reliable computing resources that would remain stable over a period of at least five years; and to making the computing resources available to users on a "free access" basis.

The time that users devote to learning about a particular computing facility and developing courseware is a valuable investment that is seriously compromised whenever the computing environment is changed. Where the computing experience of the community is modest, the prospect of change would seriously inhibit the development of computer literacy. Faculty members cannot be expected to devote their time to learning about the system and developing courseware unless they are sure that the results of their efforts will be relevant and usable over a significant period of time.

Computing resources are provided to members of the academic community on a "free access" basis, similar to library services. It is unreasonable to expect inexperienced users to estimate and justify any costs while they are learning and before the results of their efforts are known. Even when the "results" are known, it is difficult to assign meaningful dollar values to the convenience of using a computer or to the educational experiences that computer-based curriculum materials provide to students. In an environment where faculty members are not familiar with the benefits of computing to themselves and their students, a chargeback system would seriously inhibit their experimenting with the possibilities and the community's progress toward computer literacy.

The most significant decision was to centralize academic computing on a time-shared system to be purchased by the College. This had several important benefits.

a) It made it economically feasible to provide adequate computing resources with the other attributes important to the success of the project.

b) It gave us control over the operating system and courseware so that we could suit them to the particular needs of our community of users.

c) It provided a source of pride for the College community and stimulated interest in computing.

d) It helped justify the professional staff required to provide adequate technical consulting and assistance to our users.

e) It made it economically feasible to use high speed communications to make effective use of CRT terminals and provide high speed printout.

f) It made it much easier for us to collect accurate data and perform a useful evaluation of the project

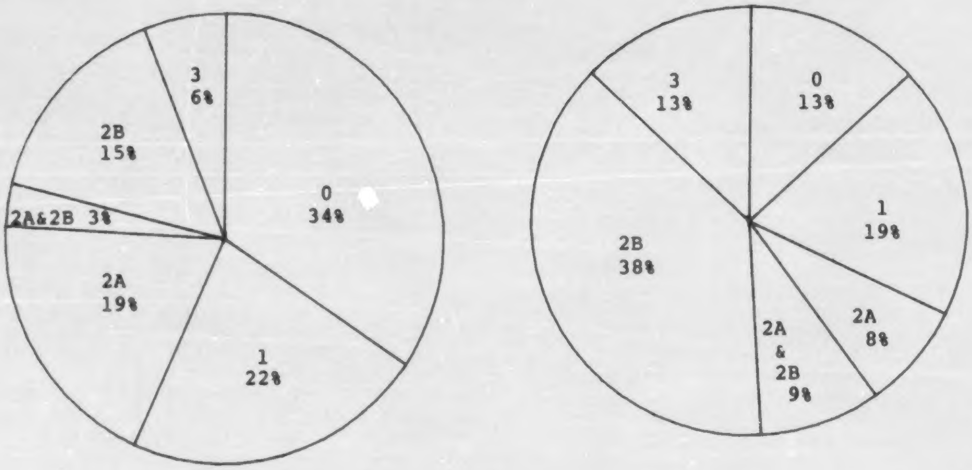
The DECsystem-20 has proved to be a good choice for our needs. It has been extremely reliable. TOPS-20 is easy for novices to learn and provides extended capabilities for more sophisticated users. We have imported many of our large software packages from other DEC-10/20 sites.

The other factors that we have identified as significant to the success of Wellesley's computer-literacy project include an effective faculty participation program; a responsive student population; and a dedicated, competent supporting staff.

Fundamentally, the success of the project is due to the efforts of faculty members to learn about computing, to develop courseware, and to use computer-based curriculum materials in their courses. Our grant from the Sloan Foundation accelerated this process by providing an attractive opportunity for faculty members to become involved in developing computer-literacy at Wellesley College. Positive student responses to computer-based assignments have encouraged faculty members in their efforts. Many students have become sophisticated computer users on their own initiative and have helped spread computer literacy among their peers. The final key to success has been a staff of competent professionals in the Computer Science Department dedicated to achieving the goals of the project.

Humanities

Social Sciences



Natural Sciences

Total Students

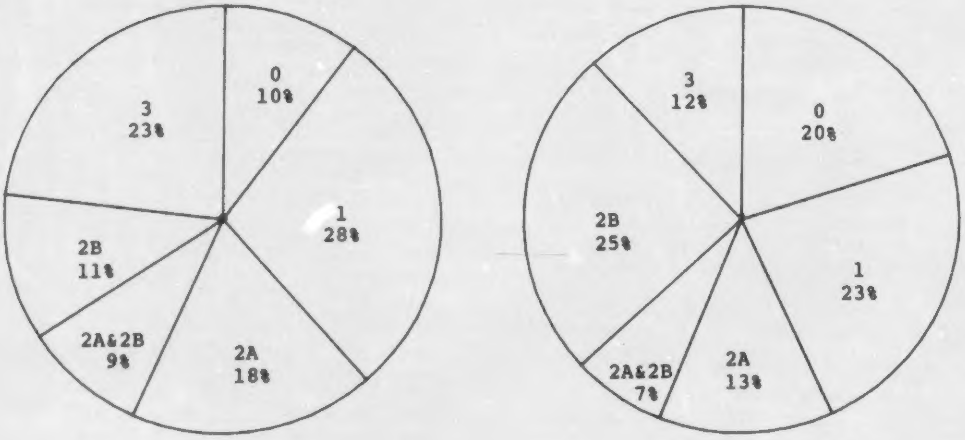


Figure 1. Computer Literacy Level Profiles of the Graduating Class of 1979.

A NOVEL APPROACH TO COMPUTER LITERACY FOR NATURAL SCIENCE STUDENTS

R.V. Rodríguez & F.D. Anger
Department of Mathematics
University of Puerto Rico
Río Piedras, Puerto Rico 00931

RATIONALE

The term "computer literacy" made its way into academic vocabulary during the past decade and has gained much currency in the last few years. Articles have been written and drums have been beaten to the effect that knowledge of the potential of computers and how to program them will soon stand along side of the three R's as an educational necessity. The computer field is so new and rapidly changing, however, that it is not clear just what should constitute computer literacy. In the recent and useful NSF report, "Instructional Computing in Minority Institutions: A Needs/Strategy Assessment," Sister Patricia Marshall uses the term restrictively to mean "the use of the computer in familiarizing students with the uses of computers and their effect on society."[2] Other proponents, including the authors, believe training should begin in primary school and eventually prepare students to use the computer as the every-day tool that it has become.[4] In this paper, however, we use the more modest objectives of the ability to read and write computer programs of moderate complexity in some high-level language and the ability to choose an appropriate computer to carry out a particular implementation. In short it means having sufficient knowledge for living and working with computers in the present computerized age.

Six years ago the necessity of offering a computer literacy course as a requirement for our natural science faculty became evident even within our insularism where the changing job demands arrive two years after USA mainland. Our students were employable then, mostly in the teaching field, but the rate of job requests had diminished while the University's main campuses had ceased to grow and many two year campuses were born.

We needed to prepare our students for present demands. We pride ourselves in offering a more solid science curriculum than most other teaching institutions in the area, and yet our students were not in demand as they used to be. With such a course we expected to offer our students a wider range of employment opportunities with better income.

ATTEMPTS, FAILURES, AND SUCCESSES

To implement our ideas we introduced a new undergraduate course, Math 107, for 3 credits which would teach PL/1 programming. At the same time we amplified our offerings in statistics for many of the same reasons. While the new statistics courses attracted few students and caused few ripples, Math 107 began with 80 students divided into two sections. Although these students were mostly selected from among the best students, only 17 stayed for the whole semester.

Why such a failure when these were the interested good students and we had put the course under two of our good professors, each with 2 assistants? We will attempt our explanation based on the firsthand experience of one of us who sat in the class that first semester and on our experience of teaching the course under its new name and approach ever since.

We should perhaps mention first, however, that the University of Puerto Rico owns an IBM/370-148 computer; six years ago we had a rented IBM/370-145 which served the Río Piedras Campus (26,000 students) and was tied with three more remote stations in other campuses. This computer is and was used for all the administrative tasks. Our faculty of 2000 students then had a remote station in which to have their cards read.

Twenty programs were assigned starting with writing the word "mama" using concatenation and using basic arithmetic operations to obtain areas up to solving n x n determinants and programming the chess problem of the knight's tour. The professors with their assistants taught the language well.

However, the computer station was not always open and the students had to wait for the research programs to run through the only printer in our faculty. There was only one key punch and the lines were long as we waited for the professors to finish. Towards the second month our station was out of cards and out of paper, while the reader and printer were broken down an average of two days each week; most programs had to run on the average of five times before a succesful outcome. Our eighty good students were carrying heavy loads in their majors which did not allow them all the hours of waiting in order to enter our tiny terminal room.

On the brighter side, that first group of seventeen which finished the course has fed our masters program in applied mathematics concentrating in data structures. Despite the drops, there were no failures, and the interest generated could be measured by the high number of those students who continued with post graduate studies.

Had we planned better, could 63 drops out of 80 students have been avoided? Would we have had the experience to create our present successful offering? Could we have forced the administration to budget the offering without the evident failure?

FUNDS, FACULTY TRAINING

A semester later we rewrote a proposal to NSF-MISIP for developing the science curriculum using the computer. The proposal was approved that summer and a frantic teaching of faculty resulted. We were to offer a course much like the Math 107 to our 2000 students, but no longer with such lack of equipment and facilities.

Where was a science faculty of 200 to find professors to teach 14 sections per semester (33 a year) of PL/1? Having lost one of the professors that had taught the PL/1 course--he was one of the earlier cases of the trend to leave academia for better remuneration in other areas--we faced the problem of hiring new faculty to teach the expanded offering in programming or training the existing faculty. With the scarcity of qualified programming instructors and our low salaries, recruiting was not easy. We were able to hire an enthusiastic numerical analyst doing research using computers, but we were unable to attract other qualified people. Thus, we had to train our own faculty.

The prospect of being trained in computer literacy was not received as a dozen roses by our faculty, but was accepted by some. (We must note that the attitude has changed to the point that now training is sought by most.) Those that did receive training were highly critical of the facilities and probably did influence the administration into accepting small classes (not our usual 30-40, but 20-25; still too high in our opinion).

Two professors and two student assistants conducted a seminar for four hours per week offering an intensive introduction to PL/1. They assigned topics to read and made the professors write related programs in the classroom. The student assistants would then keypunch and run the programs as written.

This method worked wonders. Our professors were embarrassed from their mistakes and hence did not want to show their ignorance to the student assistants; consequently they did do the assigned reading and preparation. Actually, some did, others dropped; but the ones that went on learned quite thoroughly.

Next semester each professor's schedule was carefully planned in order that he would be free at some specific hour when one of the programming courses was being offered. Thus, the professors were given more time to assure themselves before taking over the teaching of a class. When a professor was assigned to teach the course, he did a good job whether he was prepared or not. Those first classes were highly enthusiastic and well liked by the students.

In the meantime, the Biology Department requested our department to offer a statistics course for the biology majors as a requirement, but were not willing to accept more than three credits added to their required curriculum. That, of course, created a problem because we were ready to ask the faculty to approve a curriculum which already included an extra 3 credits! Since the Biology Department is our largest and we needed their approval, we began giving serious thought to their suggestion.

DIVISION OF OFFERING

The outcome was a division in our computer offering intended to better serve the needs of the different departments in our faculty. We had always recommended our statistics offering but the takers from other departments were not numerous. Now we had a chance to provide both a programming language and the needed statistics to the life scientists --biology, pre-medical, pre-dental, medical technology, pharmacy, etc., provided we could offer a well combined potpourri of both.

What about the hard scientists--the chemist, physicist, and mathematician? Should they be made to suffer the same combination of spices? Our math majors, certainly not; they should take a solid statistics course. Both chemists and physicists on the other hand should be offered more programming. Program packages in these fields are not so readily available as in statistics.

Hence our courses were to be offered in the sophomore year and at the same level for all the students of our faculty: Math 215 for biologists and premeds and Math 218 for chemists, physicists and mathematicians. Both courses have been taught mainly using PL/1 and PL/C and are designed to introduce the students to computer systems and how to make them solve problems. We intended that the students acquire a reasonable grasp of how to process numerical and character-string data using good programming practices and have some idea of the limitations and difficulties of using a large but finite machine. We also try to make them aware of the different computers they may encounter in their practice later on.

Math 215 for life scientists combines the basic features of a programming language with those of statistics. The language is taught using many basic statistics problems in median, mode, etc. as sample programs. Particularly we desire to train the student so that he may, in addition to developing his own programs, use, adapt, and change packaged programs to his needs. An example of a typical program may use DO-loops for determining the average and standard deviation of grouped data. Measures of central tendency, variability, elementary probability, normal distribution, hypothesis testing, and linear models are covered in the statistics.

Math 218 for the hard scientists includes topics such as the components of a computer system, algorithms, data types, control structures, the principles of structured programming and flow-charting, manipulation of input lists and formatting of output, arrays, subroutines, and others according to the interests of the professor and his class.

ENVIRONMENT

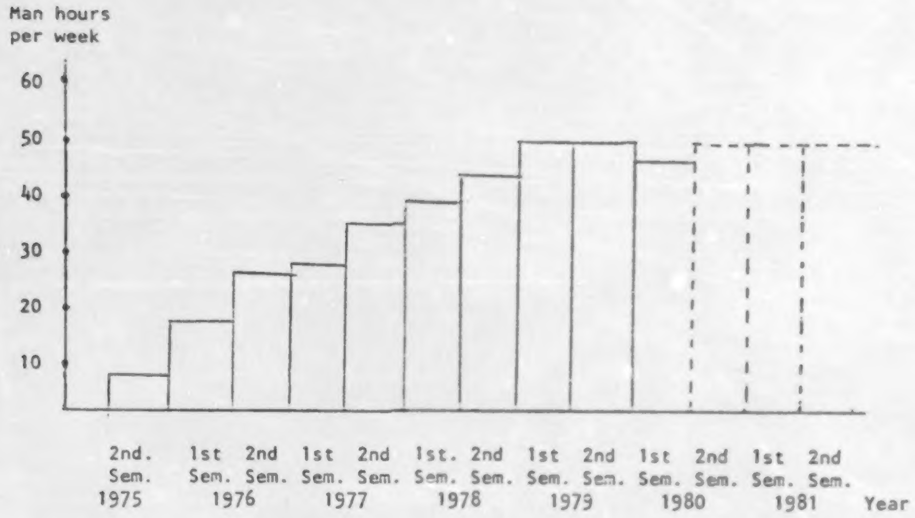
On the positive side was employing tutors, available eight hours a day to answer questions, help, and guide students in their programming. This has helped diminish the number of tries a program has to be run in order to be correct, consequently decreasing the cost, augmenting the student's confidence, and leaving to the professors only those questions that could not be answered by the tutors. If an instructor had to answer questions on every detail of the programs for each student, the hours of the day would not suffice. Hence a student may inquire of his teacher only after passing through the tutors.

However, there is an absence of assistants. The 20-25 programs have to be corrected by the professor-an unending task! Moreover there have been recent problems with the administration in maintaining the necessary space and personnel for an adequate system of tutors.

The division of the computer courses has proved satisfactory. The biology and more so the pre-medical students are very enthusiastic about their course and enjoy knowing computers, verifying hypotheses, and predicting equations. It remains to be seen how useful the course really is since a few of these first students are completing medical studies this year. It is, however, rewarding to have students come to see us specifically with the expressed motive of telling us that "PL/1 with statistics has been the most useful course I took." Those are normally the bright research students. What of the average? They seem satisfied, but more concrete evaluation is needed.

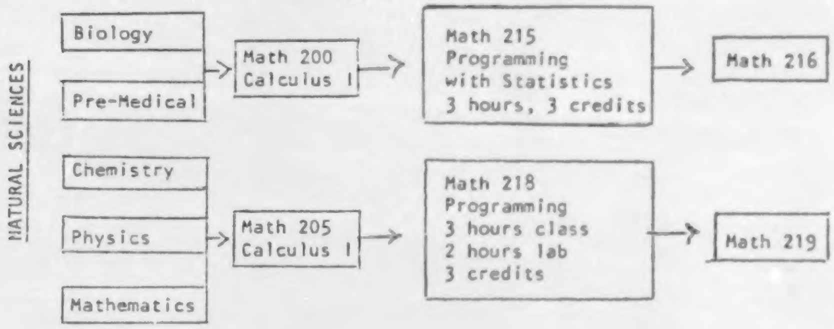
We also subdivided the faculty curriculum in core calculus courses that feed the computer courses. There is Calculus I for the life sciences (Math 200) with more emphasis on exponential functions and Calculus I for the hard sciences (Math 205), a somewhat more formal course. We described this division in our paper "A Method for Experimenting with Calculus using CAI" last year in these proceedings.(1)

FIGURE 1



Man-hours offered by the office of Programming Consultation

FIGURE 2


NATURAL SCIENCES
Biology
Pre-Medical
Chemistry
Physics
Mathematics
Math 200 Calculus I
Math 205 Calculus I
Math 215 Programming with Statistics 3 hours, 3 credits
Math 218 Programming 3 hours class 2 hours lab 3 credits
Math 216
Math 219

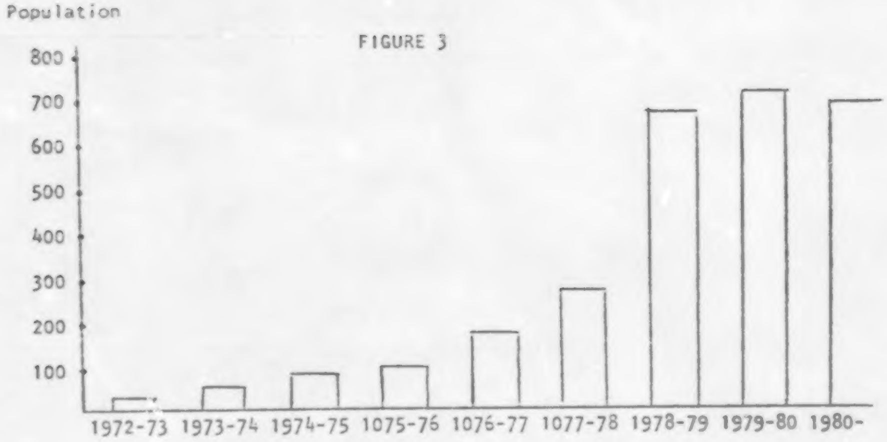
The history of our computer offering is also of interest for its ups and downs. Eight years ago we were offering a 400-level course in the Natural Science Faculty at large, called NS401. This course was taught in interactive Basic and used numerical analysis techniques. It was offered three times and many of the students that took it are occupying key positions in computer places. Not satisfied with success we changed the course to a beginner's 100-level, Math 107, already described, until it finally appears to have settled down at the sophomore or 200-level (Math 215, 218) where we think it ought to be.

ENROLLMENT IN ALL PROGRAMMING COURSES

You may question, why PL/1; why not Basic, Pascal, or Fortran? After all we are a natural science faculty and formula translator is the scientific language. This actually has been and is an ongoing debate within our faculty. There are strong reasons for all choices.

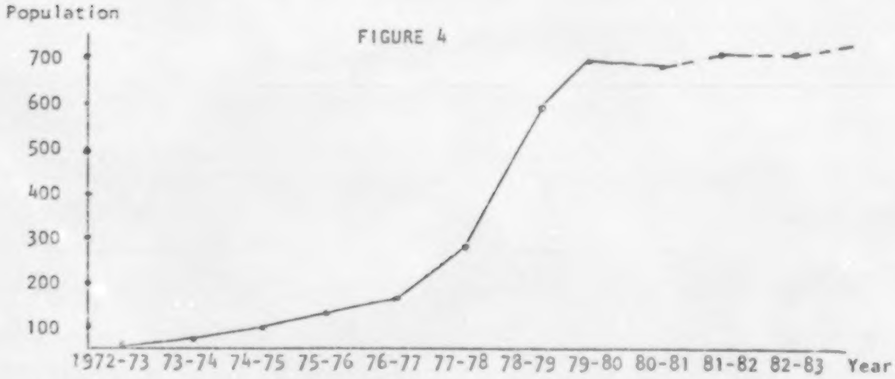
Of course, there is the problem that very few of our students will obtain jobs that will offer facilities with the PL/1 compiler available and it is true that they are more likely to find Basic. But Basic, a good beginning language, should be taught using interactive facilities which we lack. We felt we would be losing the best features of the language in batch processing. Pascal, another attractive possibility, was not so well known at the time we were making the decision, and as yet the university does not have a Pascal compiler. Fortran is the traditional standby for the sciences, but like most forms of Basic, it does not

FIGURE 3



ENROLLMENT IN ALL PROGRAMMING COURSES

FIGURE 4



Enrollment in Programming Courses with Projections toward the Future

easily lend itself to structured programming disciplines. Moreover, PL/1 combines most features of Fortran with those of Algol and Cobol to provide the widest possible range of data handling features. We felt that by learning PL/1 our students could more readily learn one of the other popular high level languages if the need arose than if they were started off with a more specialized language.

Another argument often heard is that we should offer basic knowledge of many languages, but we tend to disagree. We would rather offer our students a language that they learn well enough to develop a moderately sophisticated program. Notwithstanding, our faculty computer requirements are stated generally enough that a switch in language is easily accomplished, and so the professor is sufficiently free to choose what he considers best for his particular group.

Amazingly this liberty has not been detrimental; on the contrary, it has worked in favor of the computer literacy course. Our students pride themselves in their programming abilities and we have no programming professors antagonistic to the new course. In addition the faculty is able to adapt to innovations in programming languages, machines and pedagogy.

Our computer center, completely run by our own students, has contributed in this aspect by requiring their operators to know well PL/1 and to have taught themselves another language, mainly Basic. The interest is such that the students offer their time free to the center and will receive salary only after they have passed some difficult exams that the students themselves prepare. They consider it an honor to be a paid member of the computer center as an operator or to become a tutor for the programming courses.

Our faculty also owns a Hewlett Packard 2000 with twenty-two terminals which runs Basic and an Alpha Micro with 4 terminals. This latter machine will run Pascal but the computer is not in use yet. The H.P. is dedicated to computer-assisted instruction, thus the interactive terminals are not available for the course above described.

CURRICULUM EXPANSION

Naturally, with such drastic changes in the underclass curriculum the rest of the offerings have not remained static. As indicated in Figure 2, follow up courses were introduced to offer deeper appreciation of some of the fundamental algorithms and software development tools, including comparisons of different languages. The Math 216 course which follows the Programming with Statistics course adds to the statistical knowledge at the same time that it prepares the student to use packaged programs. Math 219 goes more extensively into overall hardware and software systems.

Besides these directly related courses, the Mathematics Department has introduced an additional four courses in computer science at the undergraduate level and three courses at the master's degree level. The department and the natural science faculty are anxious to offer a bachelor's degree in computer science, but we feel we lack sufficient qualified staff to offer a high quality program. The recent report, Instructional Computing in Minority Institutions: A Needs/Strategy Assessment, by Sister Patricia Marshall for the National Science Foundation, points out emphatically and repeatedly the great difficulties faced by any institution, and particularly minority institutions, trying to recruit computer science personnel.

EVALUATION

Our evaluation is, to a great extent, subjective but includes some hard facts and a few bitter lessons. In the first place, we learned the hard way that instituting a programming course depends on many factors beyond good intentions. Inadequate or incorrect facilities for the method chosen and number of students enrolled can lead to a situation worse than no course at all. We would very much urge interactive programming for those who can afford the appropriate equipment, but each student will need four to six times as much time at a terminal for such programming as he will need at the keypunch, which translates into many terminals. Even though we now have 10 keypunches, a 300-card-per-minute card reader, and a 600-line-per-minute printer to serve our 400 or so students per semester, we find the system very tight and, at times, intolerable!

Another factor, of course, is trained or trainable faculty, which may turn upon good salaries or good public relations. Some institutions have found that they can do with part-time teachers from local businesses what they could not afford with full-time staff. Our success depended on a few seasoned computer users, external funds for teacher training (the NSF-MISIP grant again), a little enthusiastic salesmanship, and a faculty with enough preparation and energy to throw themselves into this new challenge.

We began this paper talking about computer literacy, and any evaluation must take into account the objectives of the program being evaluated. Our intention with our basic courses was never to make programmers or computer scientists out of our students, although we count on the positive side those who afterwards turn in those directions. Success in our case is to be measured by the extent to which our students are willing and able to make intelligent use of the computing facilities at their disposal during their further studies and their careers. Most of us who teach these courses have had the experience of one of our former (or even present) students appearing on our office doorstep with a question about programming this or that project for a biology study or a chemistry lab. This is far from being a conclusive study, but it does make us feel we are on the right road. The authors know personally of at least fifteen of our former students in these basic courses who have gone on to study computers in greater depth at the graduate level and several more who are working in the data processing field, but it is too early to judge in general whether the science graduates who continue in the life or physical sciences will make much use of the skills they obtained in our courses.

We have also had our own peculiar difficulties and successes with the division we chose for the basic courses. One of the least expected but most obious difficulties has been: What do you do with the biology major that takes the required course--Math 215--and then decides to change to chemistry? He does not have the required Math 218, but you clearly cannot give him credit for both courses if he now takes the latter. We simply live with this problem, but there are enough cases to make it uncomfortable. In the success column we can count the ready approval given to the idea of these new required courses by all the departments in our natural science faculty, the positive attitute toward the course content exhibited by the vast majority of our students (once again, informally), and the relatively--for our institution--low numbers of drops and failures in these demanding courses. More concretely, Table 1 shows the number of drops and failures experienced. In our institution it is permitted to drop a course with a grade of W ("withdrawal") for almost half the semester, resulting in the Mathematics Department having an average overall drop and failure rate of 30%. In this table it can be seen that the two programming courses showed steady improvement in this area and have been for two years well below this average.

Year	Course	Total Enrollment	Number of Drops/Failures	Per Cent Drops/Failures
1971-72	NS401	4	0	0%
1972-73	NS401	30	9	30%
1973-74	none	-	-	-
1974-75	M 107	80	63	79%
1975-76	M 107	64	32	50%
1976-77	M 107	135	60	44%
1977-78	M 215 M 218	123 83	57 37	46% 45%
1978-79	M 215 M 218	435 195	71 51	16% 26%
1979-80	M 215 M 218	455 176	61 43	13% 24%

TABLE 1

RECOMMENDATIONS

When would we recommend following an offering similar to ours? Please keep in mind that we are a natural science faculty. The blend of statistics and programming formed a natural pair for the necessities of our life scientists and would probably mix agreeably under most circumstances. We do think that offering both a statistics and a computer literacy course may be preferable for more depth, but if the curriculum is already crowded, why not combine both?

If your institution lacks the funds for interactive facilities, but owns, borrows, or may use a large computer, by all means choose a programming language powerful enough that the student will eventually be confident in attacking a new one on his own. On the other hand, if funding is available, do go the interactive way. As individuals we enjoy response from our fellows; it is not so different with a machine.

Present a careful study and obtain

backing from the administration. If the latter is not forthcoming you may have to deal with a qualified failure, but it is likely to be temporary; eventually the administration will ask you to teach such a course or risk closing the institution. Such a lesson should indicate to them the real need for funds.

If computer literacy is not given the attention needed at your institution, the students will seek to fulfill their expectations elsewhere. Let us repeat a story we heard from Dr. Lem Jones from SSI. There was once an elevator operator who was satisfied with his job, but IBM thought he should progress and asked him to study new techniques. Nonetheless, Mr. Elevator Operator answered "Why should I when I am content?" A few years later automatic elevators were installed, and you may write the end of the story.

REFERENCES

1. F.D. Anger & R.V. Rodríguez, "A Method for Experimenting with Calculus Using CAI," Proceedings of NECC/2, ed. Diana Harris & Beth Collison, Virginia, June, 1980, pp. 171-178.
2. Sister Patricia Marshall,"Instructional Computing in Minority Institutions: A Needs/Strategy Assessment," NSF, SPI 7821515, 1980.
3. R.G. Selsby & Gómez, "On the Job Training of Students in Computer Science," Proceedings of Minority Institution Curriculum Exchange Conference, Concord, North Carolina, January 1979.
4. T. Sobol & R. Taylor, "The Scarsdale Project: Integrating Computing into the K-12 Curriculum," Proceedings of NECC/2, ed. Diana Harris & Beth Collison, Virginia, June, 1980, pp. 155-167.

Integrating Computing Literacy into Existing Curriculum:-- Some Case Studies

Rachelle S. Heller and C. Dianne Martin
Department of Computer Science
University of Maryland
College Park, Maryland

INTRODUCTION:

The introduction of computing power in the large-scale educational process has the potential of providing man with universal access and rapid synthesis of knowledge as well as providing a powerful creative tool for widening man's intellectual horizon. Educators must capture the moment and decide what direction this new capability will take, rather than allowing that responsibility to be usurped by the developers of computers and computer systems. The impact of computing on our educational process may prove to be one of the most important ethical issues to be dealt with in the next few years.

Ten years ago pioneers in the field of computer-assisted learning predicted that by 1980 all children would have access to computers in their classrooms. Budgetary constraints have prevented most school systems from obtaining the large-scale computer equipment needed to implement this goal. However, the advent of low-cost, low-energy microcomputer systems has allowed many schools to obtain computing power. There still remains a large segment of the population that has no classroom computing opportunities. Whether or not they work with computers in their classroom, today's student will encounter them daily. It is essential that this growing disparity between the haves and the have nots of computing be addressed immediately. To be functionally literate in today's world will require not only the basic reading and writing skills, but also an understanding and appreciation of the capabilities and limitations of computers. Therefore, computer-literacy materials must be developed based on the premise that computer literacy can be meaningfully addressed for all segments of our society, whether or not they have computing power.

Numerous computer-assisted learning projects, materials, and books have been developed, but they share a number of limitations. They were addressed to schools or school systems which have computers. Additionally, the plethora of computer hardware, software, and courseware frequently has been developed without careful consideration as to how it could be integrated into existing curricula.

Most teachers received their education before the computer explosion. As a result, with the exception of a few enthusiasts, they lack the background to implement existing materials in their classrooms (4). Books currently available assume that the educator is working with some kind of computer hardware. They include books on computer language, microcomputer design and selection, and games and problems to be solved on the computer (5).

In our view, computer-literacy material which is developed should be integrated into the existing curriculum (3). It should provide concise, fully illustrated information about the myths of computers, the history of computer devices and number systems, the structure of modern computers, problem-solving techniques, computer applications in daily living, career opportunities, social and ethical concerns, creative aspects of art, music, and poetry generated by computer, and a look at personal computing in the near future.

To facilitate integration of such material into the curriculum, a detailed cross-reference listing of topics and activities is needed. For example, when the existing curriculum requires the study of government, the teacher should be able to look into the cross-reference to find materials and activities showing how computers are important in local governmental operations. Such activities might include letters or visits to some government agency to see how computers are used, a study of laws which relate to computer use and abuse, or a simulation of how a computerized instant democracy opinion poll might work.

CASE STUDIES:

Here are some examples of activities we have used with children in grades 2 to 6 to introduce computer-literacy concepts into the classroom. The first activity we implemented allowed creative and artistic expression. We found that one of the best ways to stimulate children was to discover what their interests and starting attitudes were about computers. We began by asking them to write a paragraph about a popular computer-like character in fiction, TV, or movies. The purpose of this activity was to have them analyze whether the computer they chose was helpful or harmful to man.

This activity then was expanded into an art project. The children were told to construct a model of what they thought a computer looked like. The teacher provided boxes of all shapes and sizes, silver foil, bottle caps, and a variety of other materials. Our experience has shown that children are capable of building some fantastic creations, many of which incorporate the ideas of input, output, and control. The more enterprising students used battery power to add moving parts which they named "Radar Unit," "Cooling Unit," "Input," etc.

The second activity was a game which simulates the parts of a computer and has been used in grades 2 to 6. The children were instructed about various parts of the computer by discussion and by film. Following that introduction, the children acted out the various components of a computer: the input unit, the output unit, the central processing unit (CPU), the arithmetic unit and the memory. Every child can be used in this simulation since memory can be acted out by one child or by many children, each acting as one memory location.

A description of the set of commands possible in an assembly-like language called PPL (Pretend Programming Language) was given, such as READ, PRINT, LOAD, STORE, ADD, SUB, MULT, DIV, and STOP. Next a simple problem was described. Such a program took in two numbers, added them together, and printed the numbers and result. This problem was then developed into a PPL program. (Depending on the age of the children, the teacher would have the program ready or let the students develop it themselves.) Once the program and the data were ready on cards as shown, the game began.

```
  READ  12
  READ  13
  LOAD  12
  ADD   13
  STORE 14
  PRINT 12
  PRINT 13
  PRINT 14
  STOP

DATA: 20,14
```

The CPU commanded the input person to read the program, and the program was read into memory. The CPU sequentially examined each instruction in memory and caused it to be carried out. The children who were portraying the various parts of the computer moved in response to the CPU commands; the arithmetic unit did the arithmetic, the output unit wrote the answer, and so forth. After executing a simple program, this game was expanded to simulate more complicated programs and provided the children with a dynamic understanding of how a computer works.

The third case study involved problem-solving techniques. When the children in the fourth grade were reading word problems and working them out, we decided to include lessons in writing instructions and following them (2). Without being told, they were introduced to algorithmic design. The students were paired and given the task of constructing their own rules (algorithms) for making chocolate milk. Once they were satisfied with their design, they turned it in, received a different one from another pair, and were given the ingredients necessary to implement the algorithm. Many pairs soon realized the need for small precise steps toward the solution of a problem when, for example, they tried to pour milk from an unopened carton. The difficulties of imprecise algorithms were very literally understood - - the project was a bit messy but well worth the clean-up.

The next computer-literacy activity emphasized chronological, written, and oral skills for a computer history project. First the students were assigned a history time-line showing the important events in computing history, along with the events that they were currently studying in their social studies. Then they wrote a newspaper article or prepared a poster-talk about some important event or person in computing history. This assignment also could be used to learn about current computer applications.

A final example is based on one done with children in 5th and 6th grade. These children engaged in a project concerning paleography, the study of ancient handwritten manuscripts. They were supplied with over 100 manuscripts and many criteria to be judged. How were they to handle all the data?

At this point we introduced the concepts of computer applications and data bases. First we generally described data bases, second we provided skill builders in computer use (i.e., turning it on, turning it off, loading a program), and finally we applied the knowledge and skill to solving the problem of what to do with the data.

To introduce data bases, guest speakers talked about data collection, data entry, data management, privacy of data, and other information storage and retrieval issues. The children were excited to begin, and ready for skill builders. We planned a simple series to acquaint the students with the school's microcomputer, the TRS-80. They learned how to turn it on and off, how to load programs from tape and disk, and how to run and use simple programs. We allowed the students to try many computer games and available programs to acquaint them with the microcomputer as well as with data entry.

At the same time, in preparation for data handling and collection, the students prepared a simple questionnaire concerning parent resources (skills and hobbies that parents would be willing to offer to the school). They collected the data and used a data base management program (available on the TRS-80) to enter their data-base design and their data. The DBM sorted their data and reported it as the students saw fit. The school administration was delighted with the summary of parent resources.

Following that, the students were ready to organize the data collected about the old manuscripts. They used the DBM to look for statistical trends and frequencies in their quest for dates for these manuscripts.

CONCLUSION:

We feel these case studies illustrate how effectively and creatively computer-literacy concepts can be integrated into existing curriculum. All but one of the examples were independent of computer hardware. In the case study involving data base management, the computer hardware consisted of a TRS-80 microcomputer with one disk drive and a printer, which cost about $2,000. This small investment enabled that private school (grades K-12 with 700 students) to carry out administrative tasks, offer a computer science course to high school students, and provide curriculum enrichment in all grades. Projects such as the ones cited demonstrate that it will soon be possible for any teacher with the proper materials to introduce computer-literacy concepts into their curriculum and classrooms.

REFERENCES:

1. Beyer, Kathleen and David Moursund, "Elementary School Curriculum", TOPICS, Computer Education for Elementary and Secondary Schools, January 1981, Joint Issue by ACM, SIGCSE, SIGCUE, pages 28-31

2. Burns, Marilyn, "Getting Kids Ready for Computer Thinking, Thoughts for Teachers grades 4-8", The Computing Teacher, Volume 8, Number 1, pages 28-32

3. Hunter, Beverly, "Computer Literacy in Grades K-8", Human Resources Research Organization (HumRRO), presented at ADCIS, March 1981 in Atlanta, Georgia

4. Klassen, Dan, "Computer Literacy", TOPICS, Computer Education for Elementary and Secondary Schools, January 1981, Joint Issue by ACM, SIGCSE, SIGCUE, page 66-70

5. Petrakos, Pamela, "Rushing Toward Coursware", 80 Microcomputing, April 1981

6. "A Study of Computer Use and Literacy in Science Education, Highlight Report based on NSF Grant SED77-18658, Minnesota Educational Computing Consortium, August 1980

RESEARCH AND APPLICATION OF VIDEODISCS

Sponsored by ACM-SIGCUE

Chaired by Richard A. Pollak
Special Projects Division
Minnesota Educational Computing Consortium
St. Paul, MN 55113

ABSTRACT:

Alfred Bork, Educational Technology Center,
University of California, Irvine, CA 92717

Our educational system has difficulties at all levels--the quality of learning needs major improvements. Certain aspects of modern technology, the computer and interactive videodisc, show great promise in helping us meet some of these needs. This presentation reviews both the advantages and disadvantages of this new learning technology. It views not only the current situation, but also looks to the future of our learning society.

ABSTRACT:

Richard C. Brandt, Video Computer Learning
Project, The University of Utah,
Salt Lake City, UT 84112

Videodiscs are valuable for presenting audio-visual material that is either impossible or difficult to generate with a microcomputer. For example, the Video Computer Learning Project (VCLP) at the University of Utah uses video material to demonstrate the use of laboratory equipment and to provide actual examples of physical phenomena. Since the cost of creating video material is high, VCLP has concentrated on the development of "generic" videodiscs, discs containing material that can be used with a variety of computer-based lessons. The computer-generated material, including the branching structure of the lesson, is stored on a floppy disk which can be edited.

ABSTRACT:

Richard A. Pollak, Director, Special
Projects, Minnesota Educational Computing
Consortium, 2520 Broadway Drive,
St. Paul, MN 55113

The Special Projects Division of the Minnesota Educational Computing Consortium began to develop a model course using videodisc-supported computer instruction in August 1979. The project, "Personal Computers and the Home Videodisc Player: Solving Instructional Delivery Problems Created by Declining Enrollments," demonstrates that low-cost personal computers and home videodisc players can be programmed to deliver complete courses to students. We have designed and developed microcomputer programs for the Apple II and video segments for the Pioneer Laser Disk model VP1000 videodisc player. The videodisc can display conventional video segments along with the capability to automatically search for specific frames. The Apple II microcomputer, through an inexpensive, commercially available interface, automatically locates and plays specified segments.

There is tentative evidence that this videodisc-supported computer instruction will provide additional instructional opportunities for schools. Work with both videotape and videodisc-based systems has indicated the viability of interactive video for education. Presentations of the economic package to educators have met with enthusiastic responses. Future work is planned to continue this project along with new developmental work.

ABSTRACT:

Joan Sustik, CAI Lab, University of Iowa,
229 Lindquist Center, Iowa City, IA 52242

The intelligent videodisc project at the University of Iowa has demonstrated versatility of applications, ease of use and a concern for design factors in this new technology. In providing solutions and innovations for the needs of a major academic institution, information retrieval, research, direct instruction, and user-controlled informational sequences have been developed in the fine arts, health sciences psychology, and architecture.

An Apple/videodisc surtels constructed by the project allows for single screen display. The project currently uses a Thomson-CSF educational/industrial videodisc player under direct control of an Apple microcomputer which also serves as a communications link to other computer systems on campus.

COMPUTER LITERACY FOR ELEMENTARY SCHOOL TEACHERS

Sponsored by ACM-SIGCSE

Chaired by Jacques LaFrance
Oral Roberts University
Tulsa, OK 74171

ABSTRACT:

This session will explain the understanding of computers needed by present and future elementary school teachers. Inservice training (i.e., education, materials, and aids) needed by present untrained teachers and future trained teachers to introduce computer concepts in schools will also be explored. The following questions are a guide as to the kinds of issues that will be addressed by the panel:

What problems do non-trained teachers have learning to teach computer concepts?

How much math is required for a teacher to be able to teach computer concepts?

Are there any common attitudes that teachers have about using the new machines in class?

What materials are available to help the teachers?

Where is their experience with programs teaching computing concepts?

How much math and computer science background should future elementary school teachers have?

PARTICIPANTS:

Richard S. Lavine
Wolftrap School
1903 Beulah Road
Vienna, VA 22180

Thomas Sullivan
Stanford Avenue School
2832 Illinois Street
Southgate, CA 91790

Sue Talley
TIES
1925 West County Road B2
Roseville, MN 55113

Robert Taylor
Teachers College
Columbia University
New York, NY 10027

ELEMENTARY SCHOOL APPLICATIONS LOGO

Pat Lola
Coleta Lou Lewis
Theresa Overall
Henry Gorman, Jr.
Kay Murphy
Daniel H. Watt

ABSTRACT: Here Comes Logo

Pat Lola, The Lamplighter School,
11611 Inwood Road, Dallas, TX 75229

The Lamplighter School of Dallas, Texas, an independent early childhood education school, was given funds to implement a computer system, i.e., Logo, which could be manipulated and controlled by a young child. Fifty computers are now used to help children solve problems that are important and within the realm of their skills. The presentation will include the project's history and the school's philosophies, and how they complement each other.

ABSTRACT: Computer Lovers Under Age Six -- The Preschool Logo Program at the Lamplighter School

Coleta Lou Lewis, The Lamplighter School,
11611 Inwood Road, Dallas, TX 75229

Computer programs developed for preschool children attending The Lamplighter School allow students to obtain the effect of primitives by typing one key on the computer keyboard (e.g., Texas Instrument 99/4 home computer). The programs foster a relationship between computer use and daily activities: concepts, skills, and attitudes are developed that relate to real experiences, structure of the intellect, problem solving, classification, etc. Also, basic skills are reinforced: letter and numeral recognition, sounding of words, blockbuilding, and relationship of numbers.

ABSTRACT: How to Learn Logo Without Really Trying

Theresa Overall, The Lamplighter School,
11611 Inwood Road, Dallas, TX 75229

At The Lamplighter School both teachers and students are using Logo for learning purposes. There are several major modes of Logo -- talking to the turtle, a graphics drawing mode; working with sprites, a unique, colorful, dynamic graphics mode; teaching the computer, a programming mode; and editing, our own word processor -- as well as the usual computer printing and calculating capabilities. In this presentation many samples will be given of what students are doing and learning by using Logo.

ABSTRACT: Concept Learning After a Year of Logo

Henry Gorman, Jr., The Lamplighter School,
11611 Inwood Road, Dallas, TX 75229

As a part of The Lamplighter School Project, cognitive gains as a result of using classroom computers are being assessed by different measures. Measures of cognition must have standardized norms and be cautiously interpreted, but for the Lamplighter project there were no suitable control groups. Results of rule learning performance by third graders measured in September 1980 and May 1981 will be compared using the Boulder Norms of Bourne and O'Banion. The importance of finding rules for combining relevant

attributes in cognition will be briefly considered in interpreting the results.

ABSTRACT: Logo's O.K., Our Kids Are O.K. -- The Intangible Extras of the Lamplighter Logo Project

Kay Murphy, The Lamplighter School,
11611 Inwood Road, Dallas, TX 75229

By using Logo, Lamplighter teachers know students are fostering, mastering and experiencing attitudes and concepts that we can't prove, but know are there -- improvement in self-concept, peer relationships, ability to communicate, patience, and knowing that a computer is a powerful tool, but no smarter than the person at the keyboard.

T. I. Logo computers enable teachers to discover the learning styles of their students; it is a tool that starts at the student's level, moves at the student's pace, and challenges the student to teach himself new things.

ABSTRACT: Three Years of Logo in Elementary and Middle School Classrooms in Brookline, Massachusetts

Daniel H. Watt; Visiting Research Associate, MIT LOGO Group; LOGO Project Coordinator, Brookline Public Schools; Director, Computer Resource Center; Technical Education Research Centers; 8 Eliot Street; Cambridge, MA 02138

The public schools of Brookline, Massachusetts, have established a computer project, incorporating Logo activities into elementary and middle school classrooms, grades 4-8. This presentation will be a description of the project: the approach to teacher training, types of computers and curriculum materials used, and methods of student scheduling and classroom management. The main focus will be on the gradual growth of a computer culture among students and teachers, the types of activities students found most meaningful, and the gradual expansion of teacher interest and involvement in the project.

STUDENTS CONFRONT DATA BASE

by Doris Duncan Gottschalk, Ph.D., C.D.P.
and Roy Elliott, Ph.D.

California State University, Hayward
Hayward, California

INTRODUCTION

The authors share, as we imagine do most readers of our paper, a philosophy that computers have become an essential element of American businesses. Thus it is very important to fully demonstrate their capabilities in several college courses. Of particular importance is exposing students to computer applications they will be able to use ten years after graduation. Data bases are widespread in industry, yet they are not used extensively in business curricula throughout the country. Three years ago none was used at the School of Business and Economics at California State University, Hayward. We tried to construct our own makeshift data base. While in no sense a real data base, our makeshift gave the illusion to students of using a true data base. This paper describes our experiment.

THE ROOTS

One of the authors, Doris Gottschalk, teaches a course in data base processing and management. The other author, Roy Elliott, teaches managerial economics and has used data bases in consulting work for a number of years. For one consulting assignment the data base used was the National Bureau of Economic Research data bank accessed through the GENIE language.[1] For another the data base was the ECONODIE data bank accessed through the user language XSIM.[2] During the time he was using these two data bases, he did not know data base management and didn't need to. The commercial time-sharing companies furnishing the computers and the languages took care of data base management so smoothly that users were unaware of the complexity that characterizes data bases. The users of successful data bases are instead aware of the immense set of well-ordered data readily available for their use. Most are very impressed by this and find that data bases alter permanently their approach to problem solving. The authors felt that using a good data base would be a valuable experience for a business major and that managerial economics would be an appropriate course in which to conduct the experiment. Unfortunately, a top quality data base is very expensive.

DATA BASE DEFINED

Much controversy surrounds the meaning of data base. For more than a decade the term has been used loosely to describe a variety of computer applications.

Our concept of a data base is a group of totally integrated information records. The way in which the records are physically organized is independent of what constitutes a logical record. This is opposite the case of file management systems where there is a direct relationship between logical and physical records. Thus, a data base system minimizes the need for duplication of data, improves accuracy, and simplifies updating. A data base should have data management system software to connect the user, the host operating system, and the central processing unit. It should also have a communications processor to synchronize the control terminals used in the computer system. The overall structure of the entire data base is represented by a schema. The subschema is a portion of the data base viewed by a specific user. Some users must be able to add to and delete information from the data base as well as to access it randomly. In addition, users should be able to ask such ad hoc questions as "what real growth can be expected in GNP for 1981, assuming an annual inflation rate of 10%?" To get immediate responses to questions as this, and for instant updating, the data base system should be on-line.[3]

RECOGNIZING THE OPPORTUNITY

In the fall of 1977 the opportunity arose which would in time lead to this project: a letter from the National Bureau of Economic Research, Inc. of New York to one of the authors. The letter gave permission to use the NBER data bank free of charge for a long-term research project and also in teaching. A magnetic tape arrived shortly after containing the 2070 economic data series defined, checked, and endorsed by the NBER. We were slow to realize that this tape might be a partial answer to how our students might experience using a data base. Our bank of data on magnetic tape was far from an established data base. The user of a good data base need not be concerned with where the data values are stored, but simply calls by name whatever series he desires. Reading a particular series from the NBER magnetic tape, in contrast, required elaborate programs using obscure tape handling commands; it also required meticulous attention to the exact location and format of each data series. In a successful data base the equivalent of two powerful languages lies between the user and the data: a user language and a data base management language. To construct these languages for a true data base would have been a formidable task. We realized that a more simple, though limited, data bank was sufficient. Students in managerial economics spent at most 3 hours in class and 10 outside per quarter on all computer-related work. In this time students could not use more than a few segments of a data base. Students did not expect or even need a genuine one. To construct an imitation data base was a more manageable task for the two of us. By pooling resources, we found that it was, in fact, relatively easy.

MEETING THE CHALLENGES

Making our own teaching data base required a preliminary decision as to which of the available time-sharing computers we would use. Once that was decided, the task of constructing our data base divided logically into three problems. One problem was writing a user language, or more accurately, the few selected parts of a user language that our students could use. The second problem was one of managing data formats, specifically, copying needed data series from magnetic tape to data files on disk ready for student use. A third problem was writing new assignments related to exercises in the managerial economics text but ones profitably answered using some NBER data series. How these problems were solved is described in the paragraphs that follow.

COMPUTER SELECTION

Our first decision was which computer to use of the two state university time-sharing systems available to us: a Digital Equipment Corporation PDP-11 or a Control Data Corporation CYBER 174. We decided to use the DEC PDP-11 because it was easier to log on to and easier to learn to use. About 15% of the managerial economics students included in the experiment had no prior experience with computers; another 30% had previous experience but not on the computers available at Hayward. We had a strong preference for a computer the inexperienced students could master quickly. The CDC CYBER 174 is very powerful, but partly because of the many options it is more difficult to master. Several types of terminals are available at Hayward state, but most students are introduced to computer usage on Teleray terminals linked to the DEC PDP-11. Therefore, we chose to write all instructions for this system.

CREATING A USER LANGUAGE

We wished to build our user language around an already existing statistical package, preferably one that performed linear, polynomial, and power function regressions. We had two such packages available on the PDP-11: GYSTAT and COSAP. GYSTAT is a package developed by Jack Rhine, Jr. at San Francisco State University a few years ago, slightly modified and renamed. COSAP is an elaborate statistical and data management package from the University of Wisconsin, Appleton. COSAP overreacts to student mistakes and tends to be disk-bound. GYSTAT has neither drawback, so we chose to use it. We rounded out our user language with single-purpose programs written to mesh with GYSTAT and with GYSTAT data files. For example, one program displays a data series on the Teleray screen; another creates subseries for selected start and end dates; a copy program prints two copies of a data file on the line printer. All these programs were written to run swiftly and simply with few questions asked of users. Just those program commands useful for the current assignment were stored on the computer and accessed through the class account number. These single-purpose programs were our crude equivalent for some of the simple one-word commands in a data base user language. RUN COPY, for example, substituted for a command like COPY of a regular user language.

MANAGING DATA FORMATS

Data management turned out to be simple. Getting the series we wanted from the tape to disk was the most difficult phase. The problem was, however, nearly identical to that for other data bank tapes we had previously seen, namely, Compustat and International Monetary Fund. We transferred all the data series needed from the NBER data bank for an exercise to a data file stored on disk. Each exercise had its own data file in GYSTAT format. If a series were used in two exercises, we simply duplicated the series in two files. At most, two exercises were assigned at any one time; at the end of each assignment, the class account was cleared. This was the extent of our data management. It worked well. Undoubtedly, if we expanded use, something equivalent to a small data base management language would then be needed. With the limited usage up to now, however, timing assignments has been a reasonable substitute for a data management language.

DEVELOPING NEW ASSIGNMENTS

Writing exercises using NBER data series was a straightforward but necessary part of creating our data base. We tried to write exercises that would go with all three of the managerial economics texts we used during this time: Brigham and Pappas, Spencer, Seo, and Sirkim, and Henry and Haynes.[4,5,6] All exercises to date deal with either demand or forecasting. Both these areas are well-suited to the NBER series. In the future we intend to write more exercises in demand and forecasting, as well as to branch out to some of the other areas of managerial economics.

EPILOGUE

The crude data base we have described is still being developed. It has been tested mostly on the same student volunteers who helped build it; only recently have some portions been tested on an entire class. Yet two results are clear already. One is that overall time spent on computer exercises is reduced. The time spent learning data entry and on entry itself is virtually eliminated. More time is spent on analyzing results. On balance exercises are more valuable and take less time. The second result is that students get a glimpse of a successful data base, which we find particularly significant. We want students to have experience with computer tools they are likely to be using a decade after they graduate. Data bases certainly appear to be a tool that will be common in the future yet one that is neglected in the present college curricula.

REFERENCES

1. __________, GENIE Manual (Cupertino, CA, Tymshare, Inc.: 1974).
2. __________, XSIM Manual (Boston, Interactive Data: 1976).
3. Kroenke, David, Database Processing Fundamentals, Modeling, Applications (Chicago, Science Research Associates: 1977).
4. Brigham, Eugene F. and Pappas, James L,, Managerial Economics, 2nd ed. (Hinsdale, Illinois, Dryden Press: 1976).
5. Spencer, Seo, and Sirkim, Managerial Economics (Homewood, Irwin: 1978).
6. Henry and Haynes, Managerial Economics, 4th ed. (Dallas, Business Publications: 1978).

PHYSIOLOGY OF SMALL DATA PROCESSING SYSTEMS

Andrew Vazsonyi
St. Mary's University
San Antonio, TX

SUMMARY

Educating users of small data processing systems like small businessmen, professionals, or users of minicomputers presents a new challenge to computer educators. These users are not concerned with the traditional subjects of computer education like system design, programming, or data base management, but want a turnkey system. Experience shows that these users require some computer knowledge, but this knowledge is quite different from the knowledge required by sophisticated users and covered by traditional information system courses.

This article describes our experiences with an educational program oriented to these new types of users. It describes the specific objectives of such an educational program, the methodology developed, the new course initiated, the concurrent laboratories required to support this effort, and plans for future work.

BACKGROUND

The fastest growing segment of the computer industry in the 1980s will be dedicated to the small business user. Various forecasts indicate that millions of small systems will be installed as a result of dramatic improvement in hardware and software and a drop in cost.

End users have a dire need for turnkey computer systems, and many vendors claim the availability of such systems. However, experience shows that such systems cannot be installed without expensive assistance from vendors because unsophisticated users do not understand computer systems. They cannot diagnose difficulties and failures, and so cannot resume operations without assistance. So, during implementation continuous support from the vendor is required, and many users experience serious aggravation, delays, and cost overruns. After such systems are implemented, often there are serious operational difficulties. Frequent and costly support from vendors is unavoidable with currently available systems. Buyers today know these problems and are trying to take steps to get around them (*Wall Street Journal* [5]).

Vendors also are aware of these difficulties (L.B. Marienthal [3]) and are developing techniques to reduce the cost of software development (R.D. Gordon [2], *Wall Street Journal* [4]). Also efforts are under way to make computer systems friendly so that no special computer knowledge is required of small users. But this ideal aim is difficult or impossible to achieve. In fact, their troubles are on the increase and they are beginning to refuse to take their woes silently. Today there are court lawsuits against computer and software vendors demanding millions of dollars in damages to repay their losses (*Wall Street Journal* [6]).

Notwithstanding these facts, the trend for vendors is to *decrease* the support given to end users (*Business Week* [1]).

Thus there is a great need to educate unsophisticated users so they can implement and operate systems with more self-reliance and less dependence on frequent support from vendors.

It is useful to compare the quality of material available to educate and train data processing people and end users. For educating professionals there are excellent textbooks, workbooks, and other material available. In computer science, education is in a highly developed stage. Education of managers in the use of computer systems is also well developed. However, examining the material available to end users, the situation is quite different. Cookbook manuals are available to operate computers under routine conditions; but there is essentially no troubleshooting material available. The unsophisticated end user need not know programming, system design, or cost/benefit analysis, but does need an understanding of the form, structure, and function of those parts of the system which operate a data processing system. The end user needs explanations in everyday language without the confusion of specialized technical terms. The end user needs to understand that a computer functions in a predetermined manner and is not subject to redesign or modification.

The needs of the end user are not dissimilar to those of the biologist who studies living organisms without the possibility of changing them. Anatomy is primarily the art of separating the parts of an animal or plant in order to ascertain their position. The anatomy of a computer system should be known to end users so that they are cognizant of where the subroutines are in the system, where the data and files are located, and so on. But this is not enough, because the end user needs morphology which deals not only with location, but also with structure and form. The end user needs to know how programs tie together, what a data structure is, and how programs and data relate to each other. Finally, the end user needs physiology, the branch of biology dealing with the processes, activities, and phenomena incidental to and characteristic of life or living matter. To identify the object of our study we need to adapt this definition as follows: Computer system physiology is a branch of computer science, dealing with the processes, activities, and phenomena, incidental to and characteristic of computer systems.

Most of us know enough about the human body not to rush to a physician when we have a headache, prick ourselves with a needle, or have a stuffy nose. The user of the small system should know what to do when minor problems arise.

Our research and educational strategy has undergone considerable redirection by recognizing this need. We have decided to explain the physiology of minicomputers with the aid of inverse simulation.

INVERSE SIMULATION

It is customary to teach simulation by first introducing simulation via manual methods, and then proceeding to the computer. We have turned this approach around and decided to explain a turnkey computer system with the aid of hand simulation. Each program or subroutine is compared to a clerk who performs manual operations. Since the tasks of clerks are described in operating manuals, from the overall operational point of view there is no need to know the detailed activities of each clerk. Computer files on disks and tapes are replaced in inverse simulation by traditional accounting files. Input and output devices are again replaced by clerks and their functions are described in everyday language.

This conversion of computer operations into manual operations leads to a ready understanding of how computer systems operate. Unsophisticated users can obtain the necessary understanding to master the implementation and operation of their computer systems. To sum up, inverse simulation provides the bridge to end use for the unsophisticated user without becoming a computer expert or mastering the technical language of data processing specialists.

THE NINE PHASES OF THE RESEARCH PROGRAM

Phase 1. Perform on a computer system a case study similar to the one used in a standard educational course.

Phase 2. Dissect and analyze the hardware and software system by examining the computerization of the case study.

Phase 3. By inverse simulation create a manual model of the system.

Phase 4. Observe various system failures; examine the possibility of further difficulties.

Phase 5. Develop a general technique of diagnostics for system failure.

Phase 6. Develop recovery and restart techniques.

Phase 7. Develop pilot systems using the computer system.

Phase 8. Prepare manuals on diagnostic and recovery techniques.

Phase 9. Conduct pilot courses for unsophisticated users; test manuals and revise them if necessary.

STATUS AND PLANS

We have augmented our traditional EDP computer science course in two different ways. We initiated a special course, "Computer Software for Decision Support Systems," which carries out, with the support of specially selected students, the research program described in this article. Second, we created a series of laboratories in various functional areas of business where the computer is introduced as an integral part of the course and the students do their homework on computers. The laboratory not only educates students, but provides a test ground for faculty self-development in using computers in business education.

CONCLUSIONS AND BENEFITS

There exists a severe bottleneck in the growth of computer systems for small businesses because of the lack of understanding of unsophisticated users. We know that the knowledge required by small users is different from what is ordinarily taught in data processing and have learned how much new and required knowledge can be created. By borrowing from techniques of biology and using the inverse simulation technique, I found a promising approach to educate small users.

The research program has already shown its value to students, faculty, and users of small computers. Students find that working with real systems speeds learning. In addition, students develop a user orientation which is almost impossible to achieve in the traditional teaching of computer science. Users of small computers appear to be quite willing to work with students on their problems, and the faculty finds it effective to teach students with real computer experimentation.

REFERENCES

[1] Business Week, "Information Processing, Word Processing, Service: A Vanishing Commodity," Nov. 24, 1980, pp. 104-106.

[2] Gordon, R.D., "The Modular Application Customizing System," IBM Systems Journal, Vol. 19, No. 4, 1980, pp. 521-541.

[3] Marienthal, L.B., "Selling Small Business Systems," Datamation, Vol. 24, No. 10, Oct. 1978, pp. 86-90.

[4] Wall Street Journal, "Calculated Move. Computer Makers Feel Key to Sales Edge Lies in Better Programming," Sept. 29, 1980.

[5] Wall Street Journal, "Small Business. In Buying That First Computer, Some Homework Can Be Crucial," Oct. 6, 1980.

[6] Wall Street Journal, "Angry Computer Users Sue Burroughs for Snags," Oct. 17, 1980.

COMPUTER-SIMULATED
MANAGEMENT TRAINING
BETTER THAN
CASE STUDIES

Chauncey Burke
Robert E. Callahan
Albers School of Business
Seattle University

Our focus here, and the mission of business schools in general, is to develop students for top-management responsibilities. We business educators intend for our management graduates to integrate their topical knowledge of economics, accounting, production, marketing, and finance into countless administrative decisions. It is imperative that, somehow, the divergent business school curriculum converge to produce students able to integrate knowledge, students who will graduate with the holistic concepts necessary to function in the real world.

This goal had its origin in the Gordon-Howell report of 1959, which has been firmly incorporated by the AACSB in its criteria for business school curricula.[1] The Gordon-Howell report emphasizes the need for a business policy "capstone" course as part of the core curriculum. This polciy course should develop students' skills in applying knowledge. William Boulton has provided a useful analogy, distinguishing the sports participant and bystander: "It is easy to understand how to win at golf -- you just get the little ball from one hole to the next with the fewest number of strokes. The problem comes in application -- or the development of the skills of golf."[2]

Skill Development by Case Method

To develop the skills of future managers, it has long been thought that the case study method is the most effective approach. Because cases describe a real business situation, they provide a dynamic platform for discussion, analysis, and recommendations. There are four sets of skills essential for a practicing manager: strategic analysis, communication, human relations, and the ability to make decisions (constrained by time and uncertainty). An ordered sequence of case studies can sharpen analytical skills by covering a broad range of business environments that require both concrete and abstract consideration. The environment can be manipulated by changing the economic or industry components, and the nature of the considerations are a function of the relevant tangible or intangible data supporting the case.

The case method has proven to be an excellent tool for developing analytical skills. In a recent study of student satisfaction with the case method, respondents reported high satisfaction with problem identification and with improved ability of analysis, as a result of case studies.

In developing communication skills, the case method is again a successful tool. The very nature of case studies requires inductive reasoning and discussion.[4] Written assignments are generally an integral part of case courses.[5] Students and instructors have also expressed high satisfaction with the communication component of case studies.[6]

The two other essential skills necessary for successful management, human relations and decision-making ability, have little opportunity to develop in case study courses. Boulton has pointed out that human relations skills development is limited to whatever team projects may be assigned, and realistic management decision-making opportunities depend on the instructor's ability to simulate the real world in a "classroom culture."[7]

Skill Development by Computer-Simulated Training

The advantage of automated management games is that they not only develop students' analytical and communication skills but also provide ample opportunity for students to work with others and to make decisions in a realistic environment.

Analytical skills are improved, owing

to the immediate response of the business game. Students must interpret the financial data given, forecast the future environment, consider their hypothetical firm's objectives, anticipate competitors' strategies, and make decisions involving all operations of the firm. Their subsequent profit and market performance will reinforce their analytical abilities. The objective evaluation by the computer of the relative merits of the competing decisions has proven to be a strong persuasive force encouraging students to change their decision-making techniques. A poor profit performance will quickly identify a quantitatively or qualitatively inferior analysis. The direct, specific evaluation by the computer is usually superior to that provided in most case study courses, which is generally limited to a few subjective comments from the instructor.

Instructors can broaden the analysis by manipulating the gaming environment and increasing the complexity of required decisions throughout the exercise. For instance, changing the economic environment from growth to recession or introducing new financing options available to competitors will broaden the student's learning.

Developing the communication skills necessary for effective management is a prime concern of business-game instructors. The approach is similar to that used in case study courses. Specifically, students must submit written goals, objectives, and strategic plans for their firms. During the game, they will periodically update the statement of goals, objectives, and strategies. Most management games end with an oral and written presentation by each "firm," highlighting their performance and outlining possible future strategies.

The greatest weakness of the case study method is that students do not experience the effect of a poor management decision or the randomness of luck. Business simulations provide the opportunity to taste the frustration and stress of operating losses and deficit cash balances. How better to learn the cost of poor decisions than to be forced to contend with their outcome? In a simulation, students must retrench when their financial position deteriorates; in case studies, students simply move on to the next case.

Risk taking is an important aspect of management, In the real world, managers make most decisions based on incomplete information and assume some risk as to the outcome of their decisions. Similarly, in computer-simulated training, students not only practice assigning probability values to possible outcomes, but also experience the results of their decisions. In the safe environment provided by simulation, students discover their individual propensity for risk taking. We hope this learning will reduce costly errors in the real world.

Computer-simulation training provides opportunities to make decisions as part of a management team, the context within which organizational decisions are contemplated and finally executed. Teams can be developed that can represent top management decision makers. The teams must structure themselves to function effectively. Each team member must learn to speak clearly and cogently, listen attentively, and value the opinions of others. The team must decide whether everyone will have equal responsibility and power, or if a hierarchical structure will be adopted.

During simulation exercises, we have observed various methods that teams use to deal with the complexity of general management. We have also observed a close relationship between a team's performance (in terms of return on investment, market share, and profitability) and its assessment of group behaviors. Each team uses a brief questionnaire to monitor goal clarity, agreement on goals, personal commitment to team goals, group procedures for problem solving, listening skills, member confidence in one another, member opinion of group effectiveness, and the like. Those teams that focus on their behavioral indicators improve them through group discussion also improve their organizational performance indicators. In contrast, teams that have started with relatively high behavioral indicators have rested on their laurels, making no effort to improve their team's internal functioning. As their behavioral indicators decrease, their organizational performance also declines. On one occasion, the team that ranked first in performance for the first and several subsequent fiscal quarters ended up in last place at the conclusion of the simulation. When asked what happened, the team members replied, "We just didn't concern ourselves with how we could work together better, and it showed in our performance."

Numerous similar observations have led us to the conclusion that the human relationships encountered and handled by team members are as important an area for managerial skill development as are the more traditional areas.

Two Computer Simulation Methods

We have used two computer-simulation programs in management training: a variation of the Stanford Bank Simulation[8] and Top Executive.[9] The Stanford program was designed to simulate a medium-size bank (assets $700 million) in a competitive environment. Each program accommodates five 5-person teams, each team made up of members with different skills and expertise, e.g., a commercial lending officer, a branch manager, a trust officer, an operations officer, and a personnel manager. The five teams have to compete for loans and deposits as well as manage their capital position and investment portfolio. For the most part, teams are entrusted with responsibilities of growth, safety, and profitability. The objective of the simulation exercise is to force the team members to go beyond their individual areas of expertise to arrive at consensus decisions involving the complete range of bank strategies.

To mold the individual participants into cohesive management units, team-building exercises are used, such as a consensus-seeking test whereby team members compare results of individual decision making and group decision making. These exercises show how individual backgrounds and values affect decision-making approaches. Team behaviors are monitored throughout the simulation through periodic questionnaires.

The five teams begin from an identical financial position. Each team then determines its future growth and profitability by making a series of integrated decisions in an artificial environment representing the national eocnomy over a period of three years. The teams go through a two-stage process in developing their goals and strategies. Initially, they attempt to articulate qualitatively what the image of their bank should be at the end of the planning period. General policy statements are then developed to support the desired image, e.g., "The commercial loan portfolio will be managed to ensure quality by soliciting prime and high-grade creditors." Once the team members agree on the intended image, they translate specific goals into numerical objectives: return on investment, debt/equity ratio, return on assets. capital adequacy, etc.

After the goals and financial targets are established planning is completed when the teams have determined a system for monitoring performance. Most teams charted their execution of key financial strategies in relation to their objectives and re-evaluate their strategies as needed. During the bank management simulation, each team makes ten decisions spanning a period of two-and-a-half fiscal years. The teams make quarterly decisions involving loan pricing, service charges, marketing expenses, salary structure, branch maintenance and expansion, investments, purchase and sale of federal funds, Federal Reserve borrowing, payment of dividends, and issuing or retiring capital notes and stock. The teams make these decisions in an economy to simulate periods of rising and declining loan demand. The simulation ends with performance presentations by the competing banks. The teams interpret their results in light of their original objectives. In addition to evaluating their financial performance, they describe their team's organizational structure and decision-making process.

TOPEXEC is a business management simulation developed by Albert Schrieber, Professor of Management and Organization at the University of Washington.[10] In our opinion, this simulation is the most sophisticated management game available to business policy instructors. Schrieber originally designed it in 1956 and has made continuous revisions and improvements since then. The current version (Mark X) provides learning experiences of all general management functions. It is a highly flexible program and can be made more or less complex according to the wishes of the instructor.

A typical business policy course using this simulation organizes 20 students into 5 competing teams. The arrangement of the teams ensures a broad range of specializations, for instance, majors from finance, management, marketing, and operations systems will be teamed together. The initial class sessions are concerned with organizational structure and strategic planning. Readings and lectures develop concepts, and the teams select a company name, organizational structure, and goals and strategies. The teams are responsible for understanding the input requirements of the simulation, and are required to implement decisions through the computer. Failure to follow directions results in appropriate penalties, which become financial charges against their operations budget.

The model is flexible as to industry, type of operation (wholesale, retail), and even type of currency (mark, franc, sterling, etc.). To start the game, most instructors select a technical manufacturing environment and provide the students with the Standard Industry Code (SIC) identification. Decisions are made at quarterly intervals and typically involve 16 decisions over a two-month period (real time). The initial management decisions are limited to product pricing, raw material purchases, production scheduling, marketing and R&D expenditures, purchase of industry (competitor) information, buying additional plant capacity, and selling excess capacity, The teams submit estimates of dollar sales, net profits, and cash balances with their decisions, and the program will accumulate variations from estimates throughout the game for instructor evaluation.

Once the immediate analytical task is mastered, the instructor will introduce additional complexities by adding new phases in the simulation. Extensive use is made of the Wall Street Journal, since current market rates are used for all investing operations, e.g., money market rates, commodity investments, bond markets, and stock markets. Financing rates are determined by current interest rates and the relative position of the firm to a prime borrower as identified in the firm's financing proposals. These are some of the complexities that can be introduced:

Financing operations: Competing firms are allowed a line of credit that provides working capital funds. Subsequently, they must convert to a long-term debt structure. After they have mastered their cash planning techniques, the firms may use any financing arrangement. In past simulations we have presented, firms have negotiated loans from competitors with imaginative repayment structures; sale-leaseback plans have been arranged with competitors; bond debts with stock conversion features have been sold; stock issuances with varying underwriter agreements executed; and to prepare students for the current political-economic environment, government-insured loan packages have been offered.

Investing operations: Investing opportunities are so varied that firms can choose to discontinue manufacturing operations to become a financial operation. Any listed market in the Wall Street Journal and any financing or investment negotiated with competitors is possible.

Production management: Firms have many options in managing their volume and unit cost of production. They can minimize inventory levels without incurring stockouts and maximize their use of plant capacity to significantly affect their game performance. Management decisions affecting production include purchasing raw materials, implementing new production technology, scheduling multiple production shifts, managing plant expansion, warehousing operations, and subcontracting with competitors.

Marketing management: Firms can compete in any combination of wholesale, retail, or distributor operations. A balanced mix of pricing, promotion, and product enhancement strategies are of primary importance for success. Firms may sell inventory to competitors and bid for government contracts to improve their sale performance.

Tax planning: Tax implications of management decisions are an integral part of the game. Decisions affecting inventory pricing, depreciation accounting, capital investments, and financing will have varying effects on the firms tax picture. Any tax benefits available under current tax codes are available to competitors.

Legal and regulatory environment: Competing firms may merge and the program will automate the financial consolidations. However, the structure of the merger must adhere to the guidelines of federal antitrust legislation, specifically Section VII of the Clayton Act. On occasion, competitors have brought suit because of violations, and the classroom has become a court of law.

Labor contract negotiations have been role-played in the classroom. In a recent course we taught, firms negotiated individual labor contracts with student teams from a labor relations course. The labor contract terms were then incorporated into the program as operating charges over the remaining simulation period.

The Last Hurrah

Our enthusiasm for computer-simulated management training should be evident. We believe that, without question, it is an improvement over case studies. Simulations provide more comprehensive and more detailed experiences for future managers. In particular, they provide ample opportunity to develop human relations skills and to make decisions in a variety of environments.

We urge instructors in business policy to consider making computer-simulation training an integral part of their pedagogical repertoire, being convinced that this teaching method will play in Peoria as well as in Philadelphia. The realism and variety possible with simulations not only expand the information base for students in management, but elicit enthusiasm in students and stimulate creative thinking.

References

1. Gordon, Robert A.; & Howell, James E. Higher education for business. New York: Columbia University Press, 1959.

2. Boulton, William. Teaching case method. Working paper, Department of Management, University of Georgia (Athens), 1980.

3. Hawkins, Robert. The relationship between student and staff satisfaction with the case study method in business education. Unpublished MBA paper, Seattle University, 1980.

4. Hatcher, J. The case method: Philosophy and concepts. Boston: Intercollegiate Case Clearing House, no date.

5. The usefulness of the case method for training in administration. In Malcolm McNair (Ed.), The case method at the Harvard Business Business School. New York: McGraw-Hill, 1954.

6. Hawkins, idem.

7. Boulton, idem.

8. Robichek, Alexander; & O'Meara John, Jr. Stanford bank management simulator, version V. Graduate School of Business, Stanford University, 1975.

9. Schrieber, Albert N. TOPEXEC. Seattle: Ovoid House, 1980.

10. Schrieber, ibid.

THE ROLE OF A COMPUTER-BASED UNIVERSITY-WIDE TESTING SERVICE TO MANAGE LARGE ENROLLMENT COURSES

Lewis J. Wood
Jacqueline Ortega
Brigham Young Univ.

At Brigham Young University, the computer-based university-wide testing service serves as a major resource in the development of large enrollment classes. In fact, it is indispensible for solving problems encountered in the growth and progress of such classes. This paper describes the computer-based testing system on this campus. Then, using as examples two mathematics programs, we will discuss in detail how this system can relate to course testing and management for very large enrollment courses. Since these courses have a combined enrollment of about 4,000 students who write nearly 35,000 tests per semester, they have provided both the respective coordinators and the testing system with ample problems and situations for research and development.

DEVELOPMENT AND GENERAL DESCRIPTION OF SCOUT

Development

Testing Services (TST) at Brigham Young University (BYU) was organized in the late 1950s to provide psychological and vocational test support services to the Counseling Center. By 1966, the system was providing general test scoring services to the entire university. In 1971, the developers of this system felt that if faculty members could have the option of testing their students outside of normal lecture hours, more hours could be devoted to instruction and so began the service of providing out-of-class, computer-assisted testing.

TST management, with University administration support, began a systems analysis which culminated in developing an entirely new method of test processing at BYU--Scout (System for Computer-On-line University-wide Testing).

Hardware Configuration

Scout is supported by a 64K, 50 megabyte Microdata minicomputer. Attached to the mini are ten Adds video display terminals each equipped with a Monarch 2246 barcode reader. An NCS 7001 optical scanner is backed up, off-line, by an NCS 7005 optical scanner. Completing the system are a Centronics 6600 upper/lowercase line printer, a Printronix 300 l pm matrix printer, and a 9-track, 800 bpi tape drive. Hardware costs totaled approximately $100,000. Scout is written in an extended version of Basic. The system took about twelve months to design, code, debug, and put into operation.

Staff, Budget, and Volume

The Testing Center is housed on the main floor of the University's Harold B. Lee Library. It occupies approximately 6,000 square feet of floor space. It is staffed by three full-time administrative/staff personnel and 25-30 part-time student employees. The facility is open 12 hours per day Monday-Friday and four hours on Saturday.

The Center operates on an annual $200,000 budget and is self-supporting: academic departments and students are charged for tests administered.

Since implementing Scout on August 26, 1976, over 1,200,000 tests have been administered. The current volume is more than 300,000 tests per year, serving 260 different classes ranging in size from 25 to 2,500 students.

GENERAL DESCRIPTION OF SERVICES

There are several standard services offered by the Testing Center, variations of which can be chosen by program coordinators or instructors for their classes.

Class Initialization

Class Initialization creates a class roll for each class requesting testing services. At the beginning of a semester, before the first student comes into the Testing Center, the course instructor or coordinator submits to Testing Services a list of testing periods and test weight information, telling Scout how a particular test is to count relative to all other tests given for the course during the semester. The instructor may request a retake option. If students are permitted to retake an examination, the instructor decides which score will count in assigning a grade (either the last score or the best score). Instructors can have Scout assign letter grades, penalize incorrect answers, subtract a penalty for late tests, etc. The instructor also decides what type of test feedback information is to be reported to the student.

The first step in creating a file is to use the University's "Student Information System" on BYU's IBM 4341 computer and obtain a tape of all students currently enrolled at the University and a list of each student's current class registration. The tape is then used to initialize Scout class files. A student who enrolls late and is not included on the class roll when it is tapped can be added when he takes the first test. The class tap eliminates ID errors, speeds up student entry at the first of the semester, and gives course personnel an accurate class list. As soon as the tap is completed, Scout is ready for the class members to begin their testing.

Student Entry

During the indicated testing period, the student comes into the Testing Center. Using the Monarch 2246, a barcode label on the back of the student's University ID is read and Scout responds, indicating that the student is identified.

The Testing Center employee scans the appropriate barcode label in the test identification book telling Scout which test the student wants, and the system responds by randomly assigning a form of test to be administered. The appropriate examination is then issued. At this time, before entry to the testing area is cleared, Scout automatically checks the student's files for class enrollment. In the case of a retake exam, Scout assesses the appropriate fee and then checks to make sure the assigned form of the test has not been previously taken. The test deadline is also verified to assure that that the student is taking the test within the authorized time period, and if not, a penalty for late tests can be imposed or entry can be denied. A hold file is checked to see if the student's file is flagged, and Scout refuses student entry until all the flags are cleared. These flags can be activated for various reasons--money owed, messages from the instructor, violation of the University's regulations, etc. TST management can also flag a student if any problems exist on his cumulative test file.

As soon as entry is cleared, the student with his exam and answer sheet is admitted to the testing area. This entry and clearance process normally takes from six to fifteen seconds, depending on the student and the test he is taking. Once the student is inside the testing area, Scout allows monitoring so that management has a count of all students in the system as well as name, ID, test taken, and all other particulars about each student.

Test Scoring and Student Reports

When the student has finished his exam, he brings it to the scoring table. Here a proctor puts the answer sheet into an optical scanner. The answer sheet barcode number as well as student answers are transmitted to the Microdata for scoring. Scout's scoring program identifies the student from the answer sheet ID, selects the appropriate key to score the test, and then posts the results in the student's cumulative file. The program also transmits to the printer the student's name, social security number, raw score, percent score, letter grade, class information, etc. At the instructor's option, the program can also provide additional information such as: a list of the items the student missed, the student's responses to the questions, and the correct answers. It can also compute specific area scores within the test, thus indicating explicit areas of student weakness. Scoring, posting, and reporting takes about twelve seconds. The student exits with a hard copy of his exam results.

If the student loses his report, or if he wants a cumulative report, Scout can reprint his original report or produce a report listing all tests taken for all his classes on the system.

On-line Cumulative Faculty Reports

As soon as Scout's scoring program has posted the score to the student's cumulative file, this file can be reported to the instructor. Generally, but not necessarily, these faculty cumulative reports run at night after the office is closed and are then ready the next morning. These reports can be alphabetized by class section or overall (all sections combined). They can also be printed in social security number order without names for posting.

The cumulative faculty report lists percent and total points and Scout can assign a cumulative letter grade in addition to listing student scores on each exam. For each exam, mean scores, standard deviations, medians, modes, and scoring ranges are printed. Depending upon the option chosen when the class was initialized, the reported score on a particular exam can be either the student's best or his last score. The instructor can even change this option and get reports both ways if desired, thus allowing measurement of student progress on retake exams.

In addition to the grading features just listed, an item analysis is provided on each test item giving frequency of response, percentage of students answering each response, and a coefficient of selective efficiency to determine item discrimination. Score distribution means, modes, medians, standard deviations, kurtosis, skew, and other statistical measures are also provided to facilitate a more complete analysis of each test.

Grades Transmission

Traditionally at BYU, instructors have had the responsibility of assigning a student's semester grade, coding a grade roll, and submitting that grade roll to the Records Office. That office in turn brings the rolls to Testing Services to be optically scanned and sent to Computer Services for processing and posting to the student's transcript. Since Testing Services already had the Scout test data in machine-readable form at the end of the semester, a new module was added to the system at the Math Department's request. This module allows instructors to transmit their grades directly, bypassing the somewhat cumbersome and error-prone grade roll procedure and saving the instructor many hours of tedious semester-end work.

TESTING SERVICES AND THE GROWTH OF TWO LARGE-ENROLLMENT MATH COURSES

Two math courses make up roughly 33% of Testing Center business. These courses began to experience phenomenal growth about five years ago. As enrollment grew, the management systems for these courses began to be strained. These two courses presently serve about 4,500 students per semester. Most of the students are pursuing majors outside mathematics. Few tenured faculty are involved at this level, and a small graduate program in math does not provide enough graduate students to serve the needs of such large enrollment courses. Special management designs were needed because of the large enrollments and instructor limitation. These designs led to creating some flexible, but very stable, systems in which instructors can enter and exit without destroying the continuity of student progress from one semester to another. The instructor's concern is with content and presentation rather than course management, testing, etc.

The courses served by these systems are Basic Math Review, which is a remedial course preparatory to the Basic Math Skills Evaluation required as a part of the University's General Education program; and College Algebra/Trig, which is a calculus preparatory course, but is also terminal for many. For both of these programs, we chose mastery learning format multiple form testing, which gives feedback after each test attempt. The students are strictly paced as they proceed through the course material. Grades in both courses are assigned against fixed scales.

Basic Math Review - Organization of the Course

Math 100D covers the basic mathematical skills and applications needed to pass the General Education Evaluation (known hereafter as the G.E. exam). The course provides students with:

1. a review of applied arithmetic skills involving percentages and statistical graphs.
2. a review of common geometric figures and measurement formulas and an introduction to the metric system.
3. an introduction to basic principles and skills in working with algebraic expressions and equations.
4. experience in applying arithmetic and algebraic skills to interpreting and solving verbal problems.

This course heavily emphasizes verbal problem-solving skills, consistent with the goals of the general education experience.

The course material is organized into eight study units, each of which is followed by an examination. In addition, the G.E. examination serves as a comprehensive final.

For most students in this course, there are no lectures given and no formal class period assigned. Instead, the high-support component of this course, the math lab, offers students support and resources to meet the needs of their individual learning process.

College Algebra/Trig - Organization of the Course

Math 110 is precalculus mathematics. It contains material often described as college algebra and trigonometry, plus some additional topics. It is organized into eight individually taught and graded mini courses of *modules*:

100F	A Review of Fundamentals	1.0 hrs
110A	Polynomial and Rational Functions	1.0 hrs
110B	Exponential and Logarithmic Functions	0.5 hrs
110C	Matrices and Systems of Linear Equations	0.5 hrs
110D	Combinatorics	0.5 hrs
110E	Linear Programming	0.5 hrs
111A	Trigonometry	2.0 hrs
111B	Coordinate Geometry	0.5 hrs

A specific sequence of modules arranged for presentation during the semester at a fixed hour in a given room is called a *track*. A *track schedule*, a description of the current semester's offering of modules listed in tracks, is available to each student and is in the math lab.

A student registers for Math 110 for a variable credit of at least two hours (unless permission is given by the program director). After attending a first-day orientation, he determines which modules in which tracks he wishes to attend. The student indicates this program through an *initialization procedure* during the first days of the semester. If the student later wishes to change this initial program, he may do so up to the date posted for *finalization* of all Math 110 programs. Note that at no time may a student drop Math 110 without going through the official University drop procedure, nor may a student change his program to *less* than two credit hours in Math 110.

A student's grade in a particular module is determined by the score attained on an examination covering the material in that module. The student has the opportunity to be reexamined twice over the module at his expense, but all examinations over *a particular module in a particular track* must be completed by the date posted in the current track schedule.

SPECIFIC COURSE PROBLEMS AND CONCERNS

MATH 100D

Enrollment

Early in the development of the Math 100D program, students enrolled in the course through regular registration, but enrolled at the Testing Center by taking their first exam. This double registration resulted in some differences between the University class roll and the roll reported from the Testing Center. In addition, the testing roll also continued to carry the names of students who had dropped the class after taking one or more exams. By the end of the semester, these differences confused and concerned the instructors. Implementation of the class tap procedures in 1978 provided instructors not only with current student progress, but also with current course registration information that was consistent with University rolls.

Testing

All testing for Math 100D takes place in the Testing Center. Thus the student can choose a testing time when he is prepared and when it is convenient to spend the necessary time. The program provided by Scout paces the students through nine exams according to a schedule provided by the course coordinator. The Scout system keeps track of who has tested and how many times each student has tried each exam, and records the results of each student's best try for instructor inspection. Since about 17,000 tests are processed each semester for Math 100D, this tracking and recording service saves hours of instructor time which can then be used for tutorial, counseling, or course development.

In addition to the giant bookkeeping assistance, however, the Testing Center director and his staff have played a major roll in helping the course developers and coordinators to streamline the testing process so that students can test as quickly and smoothly as possible. Their suggestions and assistance in problem solving have been major keys to the success of system changes.

Grading

Early in the history of Math 100D, the computer assigned all grades except those altered by instructor decision. Recording final grades for submission to Records was done by hand marking grade rolls. This process was laborious and exceptionally prone to human errors. Incorporation of the grade-transmission feature of Scout relieved both the labor and the errors of the grading process. With Math 100D enrollment at about 1,500 students per semester, the relatively error-free submission of grades saves an enormous amount of beginning-of-semester grade change hassle.

Evaluation

One of the strongest developmental supports offered by the Scout system and the Testing Center is its ability to provide data and data interpretation for program evaluation purposes. Math 100D is comprised of students with a wide range of abilities and special needs. In order to serve them well, areas of weakness in the system must be identified and strengthened. Scout has provided information about the relative difficulty of our exams. It has helped to identify bad questions or questioning techniques. The system has assisted in surveying student attitudes toward a particular part of the management system or study materials. Student information such as the relationship between ACT scores and student success has helped in our counseling efforts. Together, the Center and course coordinator have identified and developed tentative solution possibilities with great regularity.

MATH 110

Enrollment

Because of the special nature of Math 110 enrollment, computer assistance is extremely helpful. In Math 110, 2,400 students enroll through regular registration. These students are then distributed into 10,000 modules which translates to a 10,000 student load by the process of initialization. The Testing Center scans and records all the initialization forms. Later in the term when each student has stabilized his program, he can retain his initial program or he may change his program by finalizing. Again, the Testing Center scans and records this new information into a final program roll which is then submitted to registration for printing of grade rolls.

Testing

Students in Math 110 accumulate about 15,000 tests during a semester: 10,000 first takes of each module final and about 5,000 retakes. Scout keeps track of student testing, paces the student according to testing intervals, and allows him within these intervals to try each exam three times. A special feature of the Math 110 program is that it actually paces the student with two sets of deadlines--an initial or bonus period and a final deadline. If the student takes and passes the test on or before the initial deadline, he receives a half-grade step bonus (B to B+, etc.) for that test. This bonus then follows the student on any retake of that exam he chooses to write. The student may also change his testing date deadlines by changing tracks. Scout contains student track information from initialization so that a terminal operator can change the track.

Grading

The computer chooses the highest letter grade from each module for which the student is registered and records the grade for transmission to records. Using the grade-transmission package in Scout is even more valuable in this course than for Math 100D. As was previously mentioned, even though the enrollment of Math 110 is 2,400 students, because each student is graded for an average of four modules, close to 10,000 grades are submitted. Before the direct grades transmission was used, the human error factor here was phenomenal. That problem has been reduced to a small, easy-to-handle number of grade problems.

RESULTS

The Department of Mathematics, in spite of the expense, supports the importance of continued involvement and research in the computer assisted areas of the two math systems described. Early in the mutual involvement between Math 100D - Math 110 and Testing Services, survival was the main impetus. The use of Scout centered around bookkeeping and flexible testing hours. Now that instructors have more time and energy available to look critically at their programs, the more powerful nature of the system, such as for evaluation efforts, has become a welcome part of the experience.

A STUDY OF THE EFFECTIVENESS OF COMPUTER-ASSISTED FEEDBACK FOR MODIFYING TEACHER BEHAVIOR IN AN INSERVICE SETTING

Ted S. Hasselbring, Ed.D.
Cathy L. Crossland, Ed.D.
N. C. State University

Teaching processes are complex multidimensional phenomena; thus, if improvement in teaching behavior is expected, then teachers must have a chance to study their teaching profiles and experiment with and practice effective teaching behaviors (Semmel, 1975). Historically, a large part of this change has been made possible using interaction analysis systems (Flanders, 1970; Simon & Boyer, 1970), which provide feedback about the teacher-pupil interaction in the classroom.

A number of researchers have successfully trained teachers to use specific interactive teaching behaviors and patterns in the classroom (Amidon, 1970; Carlson, 1974; Klein & Sorge, 1974). They provided the teacher with a precise record of his/her classroom interaction behavior, which was collected via an interaction analysis system. Generally, these systems classify the classroom interactions of teachers and pupils into various content categories that define what the designers deem important classroom processes. For example, Flanders (1964) focused on categories of teacher and pupil talk (e.g., praise and encouragement, questions, lecturing, and student-initiated talk), while Fink and Semmel (1971) focused on pupil off-task and teacher control behaviors. Lynch and Ames (1971) categorized the cognitive demands made by teachers in their interaction system.

These interaction analysis instruments allow for a systematic record of teaching acts, as well as scrutinizing the process of teaching by taking into account each small bit of interaction that occurs (Flanders, 1970). Interaction analysis systems provide an objective means of gathering data that can be used to understand a classroom situation or an individual's behavior. It is an organized way of looking for differences that may have positive or negative significance in a child's education. Interaction analysis systems have shown considerable promise as a basic training vehicle around which a comprehensive program for the development of interactive teaching skills can be used (Semmel & Thiagarajan, 1973).

APPLICATION OF COMPUTER TECHNOLOGY TO INSERVICE TEACHER EDUCATION

Interaction analysis systems have an intrinsic appeal to teachers. They (a) establish a set of operationally defined behavioral objectives for the teacher, (b) suggest an implicit set of training procedures for trainers, and (c) provide a set of ground rules that permit reliable measuring of a teacher's progress.

While ideally suited to the requirements of a skill-oriented training program, interaction analysis systems have limited use as operational tools for inservice training programs. They require extensive time by the trainers who, after observing and coding teacher-pupil interactions, must summarize, analyze, and subsequently present the results to the teacher. Thus, the total training process becomes tedious and time consuming. Furthermore, when using these traditional methods, many relevant dimensions of the observed pupil-teacher interaction (e.g., sequential patterns) are obscured, frequently oversimplifying the interchange between trainee and learner. Until limitations can be overcome in a cost-effective manner, the practical applications of interaction analysis methods to teacher education will remain unlikely (Deno, 1975; Semmel, 1972).

One possible solution to these limitations is exploiting computer technology (Turner, 1972). As Turner has stated (p. 20):

> Forthcoming development in computer utilization in teacher inservice education lies in the use of small, inexpensive real-time computers which are connected on-line to the classroom. One method of using this type of system is to code the behavior of the teacher in the classroom as it occurs, transmit it to the computer for virtually instant analysis, and transmit the analyzed behavior back to the teacher so that he/she has continuous feedback about his/her own instruction.

PURPOSE

This study investigated the effectiveness of using a Computer-Assisted Teacher Training System (CATTS) to modify teaching behaviors toward mentally handicapped students in a public school. The change in teacher behavior was defined by two criteria: the number and the type (positive) of interactions directed by the teacher toward the student.

METHOD

Computer-Assisted Teacher Training System

This study evaluated a teacher education system known as the Computer-Assisted Teacher Training System (CATTS) which was developed by Semmel (1968, 1972) over a span of years beginning in 1968. Semmel (1968) defined his system as "a closed-loop cybernetic system capable of producing continuous, immediate feedback of relevant teacher-pupil interaction to the trainee in the classroom, so modification of behavior can be realized through regulatory teaching moves in accordance with predetermined objectives" (p. 8).

To implement this behavioral change study, we used a modified CATTS configuration. It is evident that although the cost of computer hardware has dropped exponentially over the past several years, installing a computer system dedicated to supporting an inservice education system is still economically unfeasible for most public schools. So the study explored using an existing in-house central administration computer. The computer, owned jointly by several school districts and used primarily for accounting, provided CATTS feedback in a delayed mode. In the context of this study, delayed feedback was defined as providing interaction analysis data to the teachers anywhere from 30 minutes to three hours after the teaching sessions. In the context of human analysis of the same data, this turnaround time would be considered immediate.

The specific CATTS configuration developed for this study is shown in Figure 1. It consisted of four interdependent stations: a teaching and coding station, a data decoding station, a data analysis and storage station, and a data input and feedback station. Each of these stations is described and its function within the modified CATTS discussed below.

Teaching and Coding Station

The first station occurred wherever a teacher and pupil were interacting. A trained observer coded the interactive behavior of the two, and that coding provided the link between the events occurring in the classroom and the computer analysis of those events. Data from observation periods were gathered by the observer using a portable data entry device called a DATAMYTE. The DATAMYTE is a handheld button pad for numerically coded data entry which was connected to a Norelco cassette tape recorder. Data were stored on standard audio cassette tapes until they could be decoded and analyzed on the computer.

Data Decoding Station

The second station was composed of a DATAMYTE coupler connected to and housed near the remote computer terminal in the school. Observational data collected on the DATAMYTE were decoded by playing back the audio data tape into the coupler, resulting in the audible DATAMYTE codes being decoded and recorded as standard ASCII codes on a paper tape punched from the attached on-line computer terminal. The paper tape represented one observation session formatted for computer input.

Data Analysis and Storage Station

The third station (located approximately 15 miles from the research site), the data analysis and storage station, also served as the central administration computing facility. The IBM 360 computer received, stored, analyzed, and returned the coded observational data.

Data Input and Feedback Station

The fourth station, the interface between the classroom and the computer, consisted of a computer terminal. Here the decoded paper tape containing the observational data was read through the terminal into the computer over telephone lines, analyzed, stored, and fed back to the terminal housed in the school.

In summary, the observer/coder connected the events occurring in the classroom and the computer. Behavior in the teaching station was observed and coded by a trained individual, decoded by the DATAMYTE coupler, and transmitted via the remote computer terminal at the school to the computer facility several miles away. Within a matter of minutes, the data were analyzed, summarized, stored, and printed out on the remote terminal at the school. The summary printout provided a comprehensive analysis of all behavior coded in the classroom, which was then used as feedback by the teacher delivering instructional services to accomplish the predetermined socio-behavioral objectives for the students.

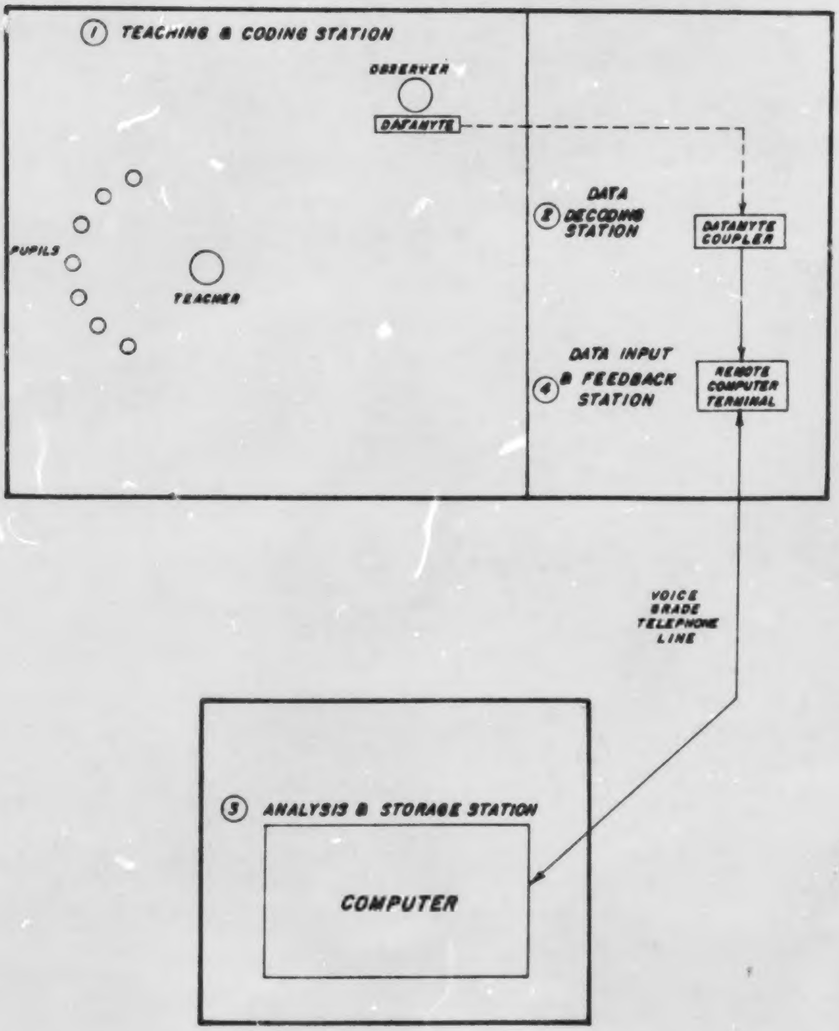
Data Collection Instrument

For this study, we had to develop an interaction analysis system to record behavioral interactions between teachers and pupils. A two-level multicategory observation instrument was designed as the primary method of data collection. The instrument was engineered to allow a trained observer to record pupil, teacher, and situational classroom data simultaneously.

The observation instrument included two levels of teacher categories: one describing the affective implications of the teacher behavior (i.e., approval, disapproval, or neutral behavior) and a second describing the type of behavior exhibited by the teacher (i.e., verbal, physical, non-verbal, etc.). Fifteen teacher categories could be generated from the eight sub-behaviors by using all possible combinations across the two levels.

Sample

The investigation was done in a public school which provided services for 150 moderately mentally retarded students and had a professional staff of 15 teachers certified in special educa-


① TEACHING & CODING STATION
OBSERVER
DATANYTE
PUPILS
TEACHER
DATA
② DECODING STATION
DATANYTE COUPLER
DATA INPUT
④ & FEEDBACK STATION
REMOTE COMPUTER TERMINAL
VOICE GRADE TELEPHONE LINE
③ ANALYSIS & STORAGE STATION
COMPUTER

tion. Nine teachers volunteered to participate in the study, which began by having the teachers assess the socio-behavioral development of their classes and state socio-behavioral goals for each of their students. These data were then used to select one student from each teacher's class who was exhibiting a need for social skill training. Thus, the target population for this study involved nine teachers and pupils.

Participating teachers were from 21 to 53 years old, with a mean age of 31. Years of teaching experience ranged from 0 to 10 years, with a mean of 3.7 years. All of the nine teachers had bachelor's degrees. Seven female and two male teachers participated in the study.

Procedures

The collection of baseline data was begun following the selection of the nine groups/duos. The baseline period consisted of a maximum number of 15 observation periods, each period lasting 20 minutes. The observer-coder collected the baseline data using the observation system. The baseline observations yielded approximately five hours of data for each duo and a total of 45 hours of data for the duration of the study, from baseline to intervention.

Following the collection of baseline data, the nine groups/duos were randomly assigned to one of three treatment conditions: (1) no CATTS feedback (control), (2) CATTS baseline summary feedback, and (3) CATTS intervention feedback.

No CATTS Feedback Condition

To begin the intervention period, the three teachers assigned to the "No Feedback" condition were each given a copy of the CATTS Teacher Training Manual and instructed to read it. The manual had three sections containing a description of the observation instrument, a discussion of the interpretation of CATTS feedback summaries, and a short instructional section on grouping and record keeping.

Following the introduction of the CATTS manual, the teachers were given the opportunity to ask questions about what they had read. The group was not, however, given any information concerning the baseline data that had been collected. The teachers were instructed to continue to work on the socio-behavioral goal they had stated for their students, but, in addition to the techniques they were already using, they were encouraged to try to be more approving of their target student and to attempt to spend more time interacting with the target student. They were reminded that the coder would continue to observe them for another 15 sessions and that, upon completion of these observations, they would be given a summary of the observations.

CATTS Baseline Summary Feedback Condition

The three teachers assigned to this group were also given a copy of the CATTS manual to read. When the manual had been reviewed, the teachers were given a computerized summary which categorized their interactions with the target pupil during the 15 baseline observations. Using the knowledge gained from the CATTS manual, the teachers reviewed the baseline summary feedback with regard to the socio-behavioral goals toward which they were working with the target pupils. The teachers were given an opportunity to pose questions regarding the baseline summary data. The teachers were then instructed to continue to work on the socio-behavioral goals for the target pupils, but to try to be more approving of the students and attempt to spend more time interacting with the students in an effort to alter their inappropriate behavior. They were also reminded that the coder would continue to observe them for another 15 sessions and that, at the end of the observations, they would receive another summary of the interaction data.

CATTS Intervention Feedback Condition

The teachers included in the "CATTS Intervention Feedback" group received the same treatment as the teachers in the "Baseline Summary Feedback" group, with one major difference. Following each of the 15 observations during the intervention period, they received a computerized summary of that session. These summaries were similar to the baseline feedback summary except that the baseline summary included the data from the 15 baseline observations while the intervention summaries represented data from only a single observation. The teachers were asked to review and record the relevant data for their behavior from each of the 15 intervention summaries using the graphing methods described in the CATTS manual. Thus, the teachers could visually determine if their behavior was changing as desired.

Experimental Design and Data Analysis

The design selected for this study was a Multiple-Group-Multiple-I design (Glass, Willson & Gottman, 1975). The design compares the three treatment groups of teacher/pupil duos with the teachers receiving differing types of feedback.

To determine the significance of the interventions, the time-series analysis technique described by Glass et al. (1975) was used. These analyses used two dependent teacher variables, the percentage of non-interactive time with the target pupil and the rate of approval interactions with the target pupil. We hypothesized that the teachers receiving CATTS interactive feedback would increase both the amount of time they interacted with the target pupils and the rate of approval interactions as defined by the observation instrument.

RESULTS

The results of the time series analysis of the three groups, as shown in Table 1, revealed that four of the six teachers in Groups I and II failed to increase significantly their amount of interaction with the target students. However, all three

Table 1
Time-Series Analysis Summary for Treatment
Groups I, II, and III
Percentage Time of Teacher Interactions
Baseline to Intervention Period

Treatment		Duo	df	t	Significance
No	I	1	28	-1.77	NS
CATTS	I	2	18	4.82	.05
Feedback	I	3	22	- .30	NS
CATTS	II	4	28	-3.79	NS
Baseline	II	5	24	2.18	.05
Summary Feedback	II	6	28	1.50	NS
CATTS	III	7	28	6.93	.05
Intervention	III	8	28	4.41	.05
Feedback	III	9	28	1.71	.05

teachers in Group III who received the "CATTS Intervention Feedback" showed a significant increase in the percentage of time that they interacted with the target pupils.

The results of the time series analysis shown in Table 2 indicate that, as hypothesized, the teachers from Groups I and II did not significantly increase their rate of positive or approval behaviors toward their target students. The teachers in Group III, however, did significantly increase their rate of approval behaviors toward their students.

Table 2
Time-Series Analysis Summary for Treatment
Groups I, II, and III
Rate of Teacher Approval
Baseline to Intervention Period

Treatment		Duo	df	t	Significance
No	I	1	28	.26	NS
CATTS	I	2	18	.65	NS
Feedback	I	3	22	1.28	NS
CATTS	II	4	28	1.09	NS
Baseline	II	5	24	-2.05	NS
Summary Feedback	II	6	28	- .03	NS
CATTS	III	7	28	4.06	.05
Intervention	III	8	28	2.09	.05
Feedback	III	9	28	4.32	.05

DISCUSSION

The present investigation has demonstrated the utility of a specific computer application for inservice teacher training. The Computer-Assisted Teacher Training System (CATTS) adapted for this study was highly effective for analyzing and feeding back interaction analysis data to teachers in an *in situ* setting.

The study further demonstrated the utility of providing teachers with continuous or ongoing inservice training via CATTS, as opposed to traditional episodic or one-shot inservice training. The results demonstrate that CATTS methodology maintains accurate data on teacher progress toward generating and demonstrating specific skills or competencies. With the increased emphasis on accountability, the maintenance of accurate records or teacher and pupil interactive behavior becomes a necessary component of inservice teacher training paradigms. Thus, CATTS methodology naturally lends itself to the demands of today's educational programs.

The dilemma now confronting both state and local education agencies is how to deliver effective training to teachers to impart the multiplicity of new skills they must quickly acquire to participate effectively in the educational process. To date, there has been a dearth of empirical data available in the area of specific and effective teacher training systems and techniques that can be used effectively. Contemporary inservice education is viewed by many as archaic, unsystematic, and ineffective. Inservice training must explore new and innovative directions. Innovative training must allow the teacher to participate actively with practice in skill generation. Training should take place in a naturalistic environment to allow for maximum skill transfer. Finally, inservice education should provide the teacher with constant formative and summative evaluation of the training activity.

Turner (1972) has espoused the use of new technology as an innovative approach to meet the needs of inservice education. He suggested that a model of inservice education should (a) use computer technology, (b) take place in the classroom during the school day, (c) be individually suited to particular classroom needs, (d) involve the teachers' active participation, and (e) give immediate feedback to the teacher on his/her performance.

Thus, the study demonstrated that it is possible to adapt computer technology to meet the needs of both students and teachers. Further, the study demonstrated that CATTS methodology could be implemented in a public school setting relatively divorced from university support structures.

The implications are significant. For the first time, teacher trainers may be able to exploit computer technology without owning and operating a computer. Moreover, only a minimum amount of technical expertise in computer technology is required. At the same time, however, the trainers can provide inservice teachers with highly specific, systematic, and continuous feedback on a large number of skills while in the natural classroom environment. Further, the use of this time-sharing network allows the teacher trainer to constantly monitor the performance progress of several teachers simultaneously.

While it was not the purpose of this investigation to establish the cost-effectiveness of CATTS, the study did effectively demonstrate the utility of CATTS as an in situ teacher training device. In doing so, it was shown that the supervisor-to-teacher ratio necessary for providing continuous in situ training was quite encouraging. During this study one observer could provide feedback for several teachers daily, whereas without the use of CATTS, such feedback would not be feasible. The introduction and rapid development of micro and mini computer systems hold great promise for realizing a cost-effective CATTS system within a few years.

REFERENCES

Amidon, E. (Ed.). Project on Student Teaching: The Effects of Teaching Interaction Analysis to Student Teachers. Philadelphia: Temple University, 1970. (U. S. Department of Health, Education, and Welfare Program 2873).

Carlson, K. W. Increasing verbal empathy as a function of feedback and instruction. Counselor Education and Supervision, 1974, 13, 208-213.

Duncan, M. J., & Biddle, B. J. The Study of Teaching. New York: Holt, Rinehart, and Winston, 1974.

Fink, A., & Semmel, M. I. Indiana Behavior Management System--II: Observers' Training Manual. Bloomington: Indiana University, Center for Innovation in Teaching the Handicapped, 1971.

Flanders, N. A. Interaction Analysis in the Classroom: A Manual for Observers. Ann Arbor: University of Michigan, School of Education, 1964.

Flanders, N. A. Analyzing Teaching Behavior. Reading, MA: Addison-Wesley, 1970.

Glass, G. V., Willson, V. L., & Gottman, J. M. Design and Analysis of Time-Series Experiments. Boulder: Colorado Associated University Press, 1975.

Klein, C., & Sorge, D. How effective is interaction analysis feedback on the verbal behavior of teachers? Educational Leadership, 1974, 32, 55-57.

Lynch, W. W., & Ames, C. Individual Cognitive Demand Schedule. Bloomington: Indiana University, Center for Innovation in Teaching the Handicapped, 1971.

Semmel, M. I. Project CATTS: The Development of a Computer-Assisted Teacher Training System. Ann Arbor: University of Michigan, Center for Research on Language and Language Behavior, 1968.

Semmel, M. I. Toward the development of a computer-assisted teacher training system (CATTS). In N. A. Flanders & G. Nuthall (Eds.), The Classroom Behavior of Teachers. Hamburg, Germany: International Review of Education, 1972, 18 (special number).

Semmel, M. I. Application of systematic classroom observation to the study and modification of pupil-teacher interactions in special education. In R. Weinberg & F. H. Wood (Eds.), Observation of Pupils and Teachers in Mainstreamed and Special Education Settings. Minneapolis: University of Minnesota, Leadership Training Institute/Special Education, 1975.

Semmel, M. I., & Thiagarajan, S. Observation system and the special education teacher. Focus on Exceptional Children, 1973, 5, 1-12.

Simon, A., & Boyer, E. (Eds.). Mirrors for Behavior II: An Anthology of Observation Instruments (Vols. A and B). Philadelphia: Classroom Interaction Newsletter, c/o Research for Better Schools, Inc., 1970.

Turner, R. L. Relationships between teachers for the real world and the elementary models programmatic themes and mechanisms payoffs, mechanisms and costs. In B. Rosner (Chairman), The Power of Competency-Based Teacher Education. Boston: Allyn and Bacon, Inc., 1972.

COMPUTING AS A WAY OF BRAINSTORMING
IN ENGLISH COMPOSITION

by

Hugh Burns, Major, USAF
Associate Professor of English
USAF Academy, CO 80840
(303) 472-3910/3930

After teaching composition for several years, I became convinced that I needed to teach more about invention, the rhetorical art of discovering what to say. In the opening of "Invention: A Topographical Survey," Richard Young describes the process this way:

> Every writer confronts the task of making sense of events in the world around him or within him—discovering ordering principles, evidence which justifies belief, information necessary for understanding—and of making what he wants to say understandable and believable to particular readers. He uses a method of invention when these processes are guided deliberately by heuristic procedures, that is, explicit plans for analyzing and searching which focus attention, guide reason, stimulate memory and encourage intuition. (p. 1)

Noble aims certainly: guiding reason, stimulating memory, and encouraging intuition. The question was *how* to reach these goals in the English composition classroom. I assumed then (and still do assume) that students frequently failed to write insightfully because they did not inquire carefully enough. I'm sure your own students use these lines too: "I don't know what to write about. What can I say about this topic anyway?" Substitute the range of topics we normally see in college composition courses and you have the problem, the challenge. Just what can you tell a student at this stage in the writing process? What could I do to spur each individual's invention process along?

While I could not hope to teach insight, I could prompt students to answer questions. I also could explain the question if the student didn't understand the basic idea or didn't know a definition. I could usually prompt a student to elaborate an answer. And I could encourage students by not rushing them and by giving them positive reinforcement if they answered the question sensibly. What I couldn't do—mentally or physically—was meet all eighty of my composition students seven times a semester for thirty to sixty minutes just to guide their initial inquiry. That's 560 conference hours teaching invention alone. Impossible. Sounded like a job for a machine. Maybe it was. Maybe the computer in the writing lab could do it. Eureka! There were enough terminals. Our university already had some CAI in basic composition. Other advantages? The computer in the lab could also print each student a copy of his or her interaction; I could not. More ideas came. I could do research; I could compare three popular heuristic strategies for their effectiveness with the computer controlling the experiment.

Everyday, I became more and more intrigued by the nature and the potential of such programs. It was soon quite clear to me that if we in the humanities must have a machine in our garden, we humanists had better create the tasks it should help us with: not a pedagogical bed of drill and practice sequences alone, but a creative, open-ended, problem-solving application of instructional computing. *Creative* in that the computer program encouraged surprises and wild connections. *Open-ended* in that the software wouldn't "know" all of the answers; it would only be programmed to move the inquiry in different directions. And *problem-solving* in that the program might sustain an investigation until some solutions were discovered.

Over the past three years, therefore, I've been designing, programming, and evaluating practical computer-assisted instruction (CAI) for stimulating rhetorical invention in English composition.

The basic idea is simple enough: a computer program asks questions, and a writer answers the questions. In terms of creativity theory, these computer programs generate content-free "matrices of thought" (Koestler, p. 38) by asking specific heuristic questions, and the writer responds with other matrices of thought based on the specific knowledge of a situational context. In other words, the computer program is responsible for the direction and the motivational sequence while the writer is responsible for the content of the inquiry. So what should happen? Arthur Koestler describes: "When two independent matrices of perception or reasoning interact with each other the result . . . is either a *collison* ending in laughter, or their *fusion* in a new intellectual synthesis, or their *confrontation* in an aesthetic experience" (Koestler, p. *45)*. This resulting interaction thus raises to the conscious level what writers already know about their subjects and makes them write down their ideas. The interaction also has an uncanny ability to produce questions that writers don't know the answers to yet. They are open-ended in that sense. Thus such dialogues, by generating some dissonance, prompt writers to articulate problems which the computer-cued interaction uncovers.

These programs have been called such names as "artificial inquiry," "electronic brainstorming," and even "Socratic Systems, Incorporated." Today, I'm happy to report that a computer can be programmed to do some, if not all, of these inventive things. Yes, this calculating engine has a place in our humanities garden as a prewriter's tool.

How do these programs work? What do they look like? Here's part of a run that one of my American literature students created answering the particle, wave, and field questions from a tagmemic invention (see Young, 1970) program. Her responses are underlined.

A COMPUTER-ASSISTED INVENTION PROGRAM: THE TAGMEMIC MATRIX

HI. WELCOME TO CAI-PROMPTED EXPLORATION.

PLEASE TYPE IN YOUR FIRST NAME: KAREN

THANK YOU, KAREN. I HOPE I CAN BE OF SOME ASSISTANCE TO YOU TODAY. IF WE TAKE EACH OTHER SERIOUSLY, I KNOW YOU'LL THINK ABOUT YOUR TOPIC AS YOU NEVER HAVE BEFORE.

BEFORE WE BEGIN, KAREN, THERE'S AN OLD SAYING ABOUT COMPUTER-ASSISTED INSTRUCTION. IT READS:

"GARBAGE IN, GARBAGE OUT!"

IN OTHER WORDS, YOU AND I MUST COOPERATE SO THAT YOU CAN GET A GOOD START ON YOUR RESEARCH PAPER.

(PRESS "RETURN" TO CONTINUE.)

WOULD YOU LIKE TO REVIEW THE DIRECTIONS AND THE COMMANDS?
(YES OR NO?)
NO

DO YOU WISH TO SEE A SHORT DESCRIPTION OF THE TAGMEMIC MATRIX?
(YES OR NO?)
YES

THE TAGMEMIC MATRIX HEURISTIC:

BRIEFLY, THE TAGMEMIC MATRIX ENCOURAGES A WRITER TO THINK ABOUT A TOPIC FROM NINE PERSPECTIVES.

FOR THIS PROGRAM, HOWEVER, I HAVE SIMPLIFIED THIS A BIT. THIS PROGRAM WILL ASK YOU QUESTIONS FROM ONLY THREE PERSPECTIVES, WHICH YOU WILL RECALL FROM OUR CLASS DISCUSSION.

1. PARTICLE—VIEWING A SUBJECT IN ITSELF (STATIC);
2. WAVE—VIEWING A SUBJECT AS IT CHANGES (DYNAMIC); AND
3. FIELD—VIEWING A SUBJECT'S RELATIONSHIP TO OTHER SUBJECTS (IN A SYSTEM).

(HIT "RETURN" TO CONTINUE.)

NOW I MUST ASK YOU WHAT YOU ARE WRITING ABOUT, SO WOULD YOU PLEASE TYPE IN YOUR SUBJECT. (I'M LOOKING FOR ONE TO THREE WORDS, MAYBE FOUR.)

DREISER'S DRAMATIC TECHNIQUE

HEY, THAT'S COOL, KAREN! WE'LL HAVE A GOOD TIME BRAINSTORMING DREISER'S DRAMATIC TECHNIQUE.

A COMMENT ABOUT PURPOSE:

WRITING WITHOUT A PURPOSE OR AIM, KAREN, IS QUITE FRANKLY A WASTE OF TIME. DOING SO GENERATES VERBAL FOG, DESTROYS WRITING EFFICIENCY, AND DEFEATS THE ESSENCE OF COMMUNICATION.

THEREFORE, THROUGHOUT THIS EXPLORATION PROCESS, YOU WILL BE ASKED TO WRITE ABOUT THE PURPOSE OF YOUR PAPER. SO NOW WOULD YOU BRIEFLY DESCRIBE THE PURPOSE OF YOUR PAPER BY COMPLETING THIS STATEMENT:

THE PURPOSE OF MY PAPER IS TO. . . .

EXPLAIN HOW DREISER USES DETAILS TO GO BEYOND MELODRAMA.

ANYTHING ELSE?

YES

WHAT?

AND TO SHOW HOW DETAILS OF CHARACTER ARE PART OF HIS DRAMATIC STRUCTURE

FINE, KAREN, YOU AND I WILL TALK AGAIN ABOUT YOUR PURPOSE.

HERE WE GO. RELAX AND ENJOY THE MIND-STRETCHING.

QUESTION 1

DESCRIBE HOW DREISER'S DRAMATIC TECHNIQUE PHYSICALLY CHANGES.

PLEASE EXPLAIN!

I'M LOOKING FOR YOUR NOTIONS ABOUT THE DEVELOPMENT OF DREISER'S DRAMATIC TECHNIQUE.

FOR EXAMPLE, IF I WERE WRITING ABOUT ANOTHER SUBJECT, SAY HUMAN DEVELOPMENT, I WOULD WRITE ABOUT GROWTH, THE CHANGES THAT ARE DUE TO MATURING. YOU KNOW ADULT "PASSAGES" HAVE BEEN POPULAR FARE LATELY.
YOUR TURN, KAREN.

I WANT TO SHOW HOW SISTER CARRIE ALLOWED DREISER TO GROW, TO CONFRONT COMMON VALUES. IT IS A MATURE WORK.

GOOD, KAREN. ADD TO YOUR RESPONSE NOW.

HURSTWOOD GROWS THE OTHER WAY. THERE ARE NEGATIVE PASSAGES, AREN'T THERE?

YES, THAT SEEMS OKAY TO ME.

ME TOO.

AH, SHORT AND SWEET. PLEASE ELABORATE A LITTLE.

WELL, CHARACTERS EITHER GET WORSE OR BETTER; THEY EITHER PASS TO A BETTER UNDERSTANDING OR NOT.

TERRIFIC, KAREN!
ANYTHING ELSE?

SOMETIMES HIS DETAILS GET IN THE WAY OF HIS ORGANIZATION AND HIS EFFECTIVENESS—BUT CONTINUE!

OKAY.

(ALL IDEAS ARE GOOD IDEAS; TYPE IN WHAT YOU THINK!!!!)

QUESTION 2—ONE OF MY FAVORITES—COMING UP:

TAKE A MENTAL PHOTOGRAPH OF DREISER'S DRAMATIC TECHNIQUE. DESCRIBE ONE IMPORTANT DETAIL.

I DON'T UNDERSTAND WHAT YOU WANT ME TO SAY

CONCENTRATE NOW ON SEEING DREISER'S DRAMATIC TECHNIQUE. WHAT FEATURE STANDS OUT THE MOST? DESCRIBE.

YOUR TURN, KAREN.

LIKE THE SETTINGS?

YES, THAT SEEMS OKAY.

THE FACTORY VS THE HOTEL.

THAT'S THE IDEA, KAREN. GIVE ME SOME MORE INFORMATION.

I SEE HURSTWOOD AS A SCAB. THOUGH HE'S BEEN REDUCED, HE STILL HAS SOME DIGNITY. BUT THE SETTINGS AREN'T DIGNIFIED. THE STREET&&

GO ON, KAREN.

THE STREETS ARE UGLY, DISMAL. CONTRAST.

SUPER, KAREN!
ANYTHING ELSE?

YES

WHAT?

I SEE SUFFERING AND COMMITMENT DESPITE IT BECAUSE HURSTWOOD HAS BOTH KINDS OF EXPERIENCE.

THAT'S AN INTERESTING REASON.
ANYTHING ELSE?

WHY DO YOU THINK THIS QUESTION IS ONE OF YOUR FAVORITES?

WHY NOT, KAREN? REMEMBER WE'RE EXPLORING FOR ANSWERS.

HOW DO YOU SEE HURSTWOOD'S SUFFERING. IS IT A PASSAGE?

I CAN'T PRETEND TO KNOW HOW, BUT YOU'VE ASKED AN IMPORTANT QUESTION. REMEMBER IT.

DO YOU WISH TO NARROW OR CHANGE YOUR TOPIC? MAYBE REVISE THE WAY IT SOUNDS IN THESE QUESTIONS?

NO.

OKAY, LET'S GO ON. . . .

Karen worked for about forty minutes. She commented twice more on her purpose as the program continued to ask ten more tagmemic questions—questions about the static, dynamic, and distributive features of Theodore Dreiser's dramatic techniques. The program told her that she elaborated (i.e., answered a question at least twice) 75 percent of the time. It also told her that she was asked four particle questions, six wave questions, and two field questions. Twice she explicitly asked for *wave* questions since she sensed that this specific, dynamic perspective was useful to her own inquiry. When she logged out, she left with eighteen pages of transcript. She was pleased.

The two other programs ask different questions. Had Karen logged on the Aristotle program, she would have been asked questions based on the twenty-eight enthymeme topics. These questions are adapted from Aristotle's *Rhetoric*, specifically Book II, Chapter 23: 1397a17-1400b35. Remembering that when Aristotle writes about invention he is most concerned with enabling us to discover the most suitable argument for persuading an audience, most of his explanations are examples of how a select topic may be applied in a certain situation. Karen, therefore, would have been asked questions such as:

> Who might believe the good consequences of Dreiser's dramatic technique are bad?
>
> Divide Dreiser's dramatic techniques into three sub-topics.
>
> What are some of the previous mistakes concerning Dreiser's dramatic technique?
>
> What is inconsistent about Dreiser's dramatic technique?

Had Karen logged on the dramatistic pentad program, she would have been asked questions based on the identification of the scene, the act, the agent, the agency, and the purpose of Dreiser's dramatic technique. Since Kenneth Burke envisions the dramatistic pentad as a more dialectical than rhetorical instrument, he traces its exploratory appeal not to Aristotle's system of topics but to Aristotle's classification of causes (Burke, p. 228). Burke's rhetoric, therefore, differs from classical rhetoric in that his major concern is not persuasion but *identification*. Four pentad questions Karen might have been asked are:

> What impresses people about the setting for Dreiser's dramatic technique?
>
> What audience would most appreciate knowing more about Dreiser's dramatic technique?
>
> What is the ultimate goal of Dreiser's dramatic technique?
>
> How is Dreiser's dramatic technique like mercury?

We all know a rhetorical renaissance has recently emerged within the teaching of English composition, but most of us aren't as ready or as eager to admit that an electronic revolution has emerged as well. What such programs illustrate above all else is that rhetorical invention and computer technology are indeed compatible; combing heuristic modes and computer media can well serve and gladly teach the inquisitive writer. I'm only echoing English educators like Ellen Nold of Stanford University when I report such heresy:

> What is preventing humanists from using the computer for humanitarian purposes is merely their belief that they cannot use the machine. It is ironic that a group known to undertake calmly and surely the study of Latin, Greek, Russian, Chinese, Swahili, or Gaelic often balks at the much simpler task of learning the more logical, far less capricious, language of the machine. (Nold, pp. 272-73)

Brainstorming in English composition through computer-assisted instruction is, I'm convinced, an effective, supplementary way to begin teaching the art of systematic inquiry. It's also a most appropriate introduction to the richness of heuristic strategies in general. Look for creative computing approaches to enrich what we humanists can do. We have a tool, but you and I must use it well. After all, if this machine is already here, shouldn't we humanists be the ones to till the garden we all labor in?

References

Aristotle. *The Rhetoric and the Poetics of Aristotle*. (W. R. Roberts, I. Bywater, trans.). New York: Modern Library, 1954.

Burke, Kenneth. *A Grammar of Motives*. Berkeley: University of California Press, 1969.

Burns, Hugh, & Culp, George. Stimulating Invention in English Composition Through Computer-Assisted Instruction. *Educational Technology*, 1980, 5-10.

Koestler, Arthur. *The Art of Creation*. New York: Macmillan, 1964.

Nold, Ellen. Fear and Trembling: The Humanist Approaches the Computer. *College Composition and Communication*, 1975, *26*, 269-273.

Young, Richard. Invention: A Topographical Survey. In G. Tate (Ed.), *Teaching Composition: Ten Bibliographical Essays*. Fort Worth: Texas Christian University Press, 1976.

Young, R. E., Becker, A. L., & Pike, K. L. *Rhetoric: Discovery and Change*. New York: Harcourt, Brace, and World, 1970.

GRAPHIC DESIGN ISSUES IN COMPUTER-BASED EDUCATION

Sponsored by ADCIS

Raymond Nichols
Jessica Weissman
Brian Shankman

ABSTRACT: Visual Design

Raymond Nichols, University of Delaware, Box 890, Newark, DE 19711

Design is to visual elements as intonation is to speaking. Two different speakers can give a talk using the same words and achieve startlingly different results. We can be bored listening to one, yet sit on the edge of our chair with the other. The key is the quality and sometimes the form of the presentation. Visual design is the method of presentation for information in computer-based instruction.

Design is the way you emphasize specific points, visually collecting similar ideas and separating others; direct the eye from one word to the next, one sentence to the next, one paragraph to another; and create a visual pause and develop rhythm. Words and pictures are the information that you want to relay to some student or observer; visual design presents that information. A word softly spoken creates an impression, yet the same word spoken harshly achieves a different response.

Visual design can evoke the proper feeling from the viewer, make him more receptive to the educational experience being transmitted, and provide him with subtle hints for organizing and retaining information.

ABSTRACT: Achieving Desired Graphics Effects

Jessica Weissman, University of Delaware, Computer-Based Instruction, Newark, DE 19711

This talk covers two topics concerned with achieving desired graphic effects.

A. General techniques for text display: arranging material so it is attractive to read; using layout and graphics to help students quickly understand the material and the directions. Material should be arranged in logical order (top to bottom, left to right). It should come on the screen in manageable chunks. Avoid crowding by using blank space for readability. Consider using different parts of the screen for different functions (text, question, feedback).

B. Using graphics for emphasis: boxes, lines, circles, and special styles or sizes of writing can help a student understand the relative importance of the components of a display. Consider how many times the student must see a display to decide which method of emphasis to use. Avoid techniques which take a long time to plot. Underlining and letterspacing are all useful techniques. Asterisks, dashes, centering, etc. can help provide emphasis, too. Finally, remember that if you emphasize everything, nothing on your display will stand out for students.

ABSTRACT: Graphic Strategies

Brian Shankman, 116 Manchester Drive, No. 216, Euless, TX 76039

Although highly detailed graphics and real-time simulations and animations are a desirable aspect of computer-based instruction, such is not always the case. At American Airlines, the accuracy in instrument readings used for pilot training demands higher resolution screens. Although high resolution graphics are needed to display individual gauge readings and dial settings, there is a tradeoff in plotting speed and the amount of memory required to map each addressable location. In addition, real-time simulations are often bypassed for instructional reasons. For example, a

pilot making a 30° turn in real time might take a minute or two before completing his turn. The advantage of using computer-based simulations is that we can show the result of the turn and individual stages of the turn as we please, allowing the student to absorb the information. For real-time simulation, we use a true motion-based simulator.

Special care must be taken when transferring three-dimensional objects to a two-dimensional screen, as students must be able to transfer between the "real thing" and its graphic representation. At American, our students, who are already experienced pilots, get hands on time in a Cockpit Procedures trainer with the actual instrument layout and working instruments. They also have color slides, adjacent to the terminal, for each instrument panel and schematics, minimizing the need for high-resolution color graphics. It is best to make the displays as simple as possible while still maintaining fidelity to the object you are representing. Character sets or plotted characters created from special graphic characters can be optimized for maximal use of memory and programming time.

EVALUATING AND SELECTING COMPUTER STUDIES TEXTBOOKS:

ART, SCIENCE, OR CHANCE?

Chaired by Stephen Mitchell

ABSTRACT:

Teachers in the computer education disciplines face unique difficulties in evaluating and selecting textbooks for their courses. In general, these difficulties are most apparent in the lower division courses, where larger enrollments, multiple instructors for the same course, and pedagogical considerations for the student affect the decision.

The following three major differences in computer education distinguish this task from other academic subject areas:

1. The impact that changing technology has on book content: assessing current versus obsolete topics. Problems for teachers and problems for publishers.
2. Programming coverage questions: choice of language. Structured versus unstructured coverage. System dependence and language standardization questions.
3. The increasing reliance on part-time instructors: implications on textbook selection and use. Distribution difficulties for publishers.

The following similarities to other subject areas in selecting books for lower division courses are acknowledged:

1. Pedagogical book features and learning aids become more important.
2. Instructional support materials have greater utility.
3. Reading level analyses become more prevalent.
4. Textbook prices may or may not become a factor.

This panel discussion session is intended to provide dialogue between publishers and educators on the above seven evaluation and selection areas, as well as any others suggested.

The question of the desirability of shared information on various more formalized rating standards, reading levels, computer currency scales, comparative price data, current user experiences, etc., will be discussed.

PARTICIPANTS:

Nina Lewis
John Wiley & Sons, Inc.
605 Third Avenue
New York, NY 10016

James Leisy
Brooks/Cole Publishing Co.
Division of Wadsworth, Inc.
555 Abrego Street
Monterey, CA 93940

Jon Thompson
Wadsworth Publishing Co.
10 Davis Drive
Belmont, CA 94002

Charles Murphy
Mayfield Publishing Co.
285 Hamilton Avenue
Palo Alto, CA 94301

Tom Walker
Boyd & Fraser Publishing Co.
309 South Willard Street
Burlington, VT 05401

STRUCTURED SYSTEMS ANALYSIS TUTORIAL

Sponsored by IEEE/Computer Society

V. Arthur Owles
Michael J. Powers
Applied Computer Science Department
Illinois State University
Normal, IL 61761

ABSTRACT:

The systems analysis process is riddled with problems, even in those organizations which employ a well-defined system development methodology.

The analysis process is typically imprecise, incomplete, and haphazard.

The communication between the analyst and the user on the one hand and the designer/programmer on the other is clumsy and inaccurate.

The new system general design prepared by the analyst is likely to be a mere pertabation of the old system design, whether that makes sense or not.

Structured design concepts and tools have been applied most heavily to the program design process. Their potential for use in the more global system design process has been lost largely because there has been no natural starting point. Further, the use of structured design tools has only widened the gap between analysis and design; the traditional product produced by an analyst is so far from the starting point of the structured designer that considerable effort is needed to translate or codify it into a useful document (typically uncovering analysis errors and omissions in the process). Worse yet, the analyst's product may simply be ignored.

Structured systems analysis provides an approach and a set of tools designed to solve these problems.

The tutorial intends:

To introduce the basic tools of structured analysis.

To introduce a structured analysis methodology.

To show the connection between structured analysis on the traditional system development life cycle.

To present recommendations concerning the teaching of structured analysis and design.

COLLEGE CAI PROJECTS IN CONTENT AREAS

Steven V. Bertsche
James D. Spain
James W. Parrish
Barron Arenson
Matthew Wozniak

ABSTRACT: Music CAI at the University of Wisconsin-Superior

Steven V. Bertsche, Music Department,
University of Wisconsin-Superior,
Superior, WI 54880

About 60 music students at the University of Wisconsin-Superior are now using 90 competency-based drill and practice lessons authored by Wisconsin and North Texas State faculty and staff. Lessons cover chordal listening, harmonic distortion, pitch patterns, fundamentals, intervals, and scale forms.

The programs allow random ordering of questions, fixed or random transposition of pitch level, and author-selectable error diagnostic routines. In addition the lessons collect data from student responses to generate a statistical summary for each lesson and weekly student performance reports.

ABSTRACT: The SUMIT Courseware Development Project

James D. Spain, Department of Biological Sciences, Michigan Technological University,
Houghton, MI 49931

The National Science Foundation's Development in Science Education Program is supporting a two-year project to develop and evaluate microcomputer-based courseware for biological science instruction at the post-secondary level. The project name is an acronym for Single-concept User-adaptable Microcomputer-based Instructional Technique, and emphasizes development of a large number of relatively small programs which are clearly written in Basic and fully documented to permit easy adaptation to local needs and interests. The courseware is being written primarily for the Apple II Plus microcomputer.

The SUMIT project has been in operation about one year and has resulted in about 20 programs, although many still require additional documentation. During the coming academic year, these programs will be evaluated in biology courses at Michigan Tech, principally General Biology and Animal Ecology. It is planned that all documented and evaluated programs will be reviewed by CONDUIT and made available for general use some time in late 1982 or early 1983. In the meantime, interested individuals may obtain copies of working programs by writing to the author.

Written descriptions of programmer's aids will be made available at the presentation of this paper. Of particular interest are the specific methods for making effective use of the Apple memory when using high resolution graphics, and the method of simultaneously graphing data on both pages of high resolution graphics. An overview of the program objectives will be presented, along with specific examples of programs that have been developed.

ABSTRACT: The Computer as a Tutor in Music Education

James W. Parrish, Music Department,
Wingate College, Wingate, NC 28174

Computerized instruction has been found to be not only feasible, but effective and efficient in many areas of the music curriculum of public schools and higher education. With the development of graphics and audio capabilities in microcomputers and certain main frame computer systems (most notably the Control Data Corporation's Plato system), instructional units in music history, acoustics, education, and especially theory are being automated.

Units of computerized instruction and drill have been designed to supplement classroom instruction in the freshman music theory classes at Wingate College. The project has used a Micro Data Reality real time interactive Basic/Cobol computer (50 million bytes of disk, 32K of virtual memory), as well as especially designed workbooks and audio tapes. Units to be reviewed include note name identification, triad identification, triad construction, and melodic dictation. The presentation will include handouts and discussion of both the off-line and computerized materials, and a slide presentation and discussion of the instructional strategies employed.

ABSTRACT: A Microcomputer Enhanced Physics

Laboratory

Barron Arenson, Physics Department, Mesa Community College, Mesa, AZ 85202

Matthew Wozniak, Physics Department, Scottsdale Community College, Scottsdale, AZ 85253

A new laboratory program for the second semester general physics (calculus level) course has been developed that emphasizes the use of microelectronics and microcomputers to measure and analyze laboratory data. The laboratory units are Basic Electrical Skills, Oscilloscope, Sound Velocity, Amplification of Experimental Signals, Analog and Digital Signals, Data Acquisition Using a Microcomputer, etc. A summary of the features of the program will be presented, and copies of the laboratory manual will be available.

Two interface cards for the Apple II microcomputer have been developed, allowing the student to easily use the computer in the gathering and analysis of experimental data. The analog card has a software-controlled eight channel multiplexed analog to digital converter and two separately addressable eight-bit digital to analog converters. The digital card has eight separately addressable single-bit inputs, eight separately addressable single-bit outputs, one eight-bit input port, and one eight-bit output port. The design features of the cards and their use in the laboratory will also be discussed.

SIMULATION OF COMPUTER ARCHITECTURE USING A HIERARCHICAL COMPUTER HARDWARE DESCRIPTION LANGUAGE (CHDL)

W. A. Skelton
R. S. Walker

ABSTRACT

This paper describes a system that assists computer architecture studies in microcomputer education courses. The Computer Hardware Description Language (CHDL) and simulator operate at the gate, register, and component levels. The system is specifically designed to work with microprogrammable architectures such as the AMD 2900, thus permitting higher level organization studies. The CHDL called CALSIM (Computer Architecture Language for SIMulation) supports a hierarchical description for the architecture of the object machine. Logical components including memories are described using a series of register descriptions, pin descriptions, and Boolean expressions. Individual wires of connectors are both numbered and named. The flow of the logic through the model is provided by wiring statements which connect the components and connectors to each other. The contents of main memory, micromemory, and one or more auxiliary memories are stored in files which are read into the system prior to simulation. CALSIM was constructed using LALR(k=1) grammar. The grammar was first processed through an analyzer. The analyzer output drives the syntactical analysis portion of the compiler which converts the hardware description input into tables. These tables not only drive the simulator but also provide the user with a tabular description of the machine design. The simulation driver design incorporates breakpoints, trace, display, and other tools to allow the user to follow the simulation, either in a single step or in a run mode. The entire system is compiled as a single package using the DEC 20 as the host machine. This paper first describes the language and software, then gives examples of the language and, finally, describes the experience of a graduate computer organization class which used the system in a class project.

INTRODUCTION

One of the difficulties in working with designs using bit slice components, such as the AMD 2900 series, is the necessity of a design-specific software simulator or an expensive emulator. Alexandrididis (1) states, "The advantages of bit slice microprocessors are not free -- they require a serious investment in system support software." The problem is critical in teaching where funds may be limited and simultaneous access to equipment by many users is essential. The Computer Science department at the University of Texas at Arlington (UTA) has used AMD 2900 and Intel 3000 micro-code assemblers for several years. Although useful, they do not allow simulation. The student needs to prepare the microcode, program the test code, and then execute the code in a real-time simulator using test data. The plan outlined below was developed in response to those needs. The system described allows the student to design a simple machine using components of his own choosing. The design can then be compiled and executed in a single software package. When this is completed successfully, the student has a working model of his hardware design. The model can be used to test various microcode, program, and data input. Portions of the compiler/simulator were used for the first time during the fall semester 1980 at UTA. Students in an introductory graduate computer organization course used the language and the syntactical analyzer to describe simple architectures of their own design. Suggestions from students have been used to revise the language, the grammar, and the compiler.

LANGUAGE AND SOFTWARE: THE PLAN AND IMPLEMENTATION

The development plan for the software compiler/simulator included the following key items:

- Design a register level hierarchical hardware description language.
- Use the LALR grammar analyzer (2) to assure a cohesive grammar.
- Develop a data structure which could be used as output from the compiler and as input to the simulator.
- Develop a compiler using the output tables from the LALR grammar analyzer to drive the syntactical analysis portion of the compiler.
- Develop the semantic portion of the compiler, which prints out descriptive tables of the hardware.
- Provide for file input and printout of the main memory, micromemory and auxiliary memories of the object machine.
- Provide file I/O to and from at least three ports during simulation.
- Provide library support that accepts component descriptions and provides copies of items in the library.
- Prepare a simulation driver which allows the user to set break points based on conditions on the data bus, address bus, and the timer setting.
- Allow the user to build menus of various displays that may be used on command or in the run mode.
- Provide for a trace during simulation.
- Construct a simulator which directly executes the user logic.
- Provide a timing system to control the execution sequence.

The project was implemented in the fall 1978 and the language was designed in the spring and summer of 1979. The top level of the compiler/simulator written in the fall 1979, routes the user to various environments--the compiler, simulator, library (to search/edit), micromemory read-in, main-memory read-in, table, and documentation requests. The lexical scanner and the syntactical analyzer of the compiler and several of the minor modules were completed in the spring of 1980. The semantic analyzer was completed during the fall semester 1980. Revisions, based on input from students (discussed later), were completed following that semester. We began working on the Simulator in spring 1981 and hope to complete the project for fall 1981.

LANGUAGE AND SOFTWARE: HOST MACHINE AND HOST LANGUAGE

Because of its greater interactive capabilities, the DEC 20 was chosen over the IBM 370. The DEC 20 not only provides user work space but also has several system commands which are active inside application software to assist the user. We chose Cobol as the host language because of its superior data organization capabilities, simple user interface, and self-documenting features.

LANGUAGE AND SOFTWARE: THE GRAMMAR AND LANGUAGE

The design requirements for the Calsim grammar are based on features of hardware description languages found in (3) through (19). These guidelines require that the grammar:

- Be hierarchical.
- Accept register-level and gate-level logic.
- Allow storage and retrieval of component descriptions in a library.
- Accept connecters, modules, and terminals as part of the hierarchy.
- Allow insertion of comments.
- Allow timing of the logical actions.

- Allow storage and retrieval of component descriptions.
- Be statement oriented.
- Recognize the unique features of memories, micromemories, auxiliary memories, and peripherals.
- Directly execute the logic of the components.

LANGUAGE AND SOFTWARE: THE TOP STRUCTURE OF THE COMPILER/SIMULATOR

Concurrent with the use of the compiler/simulator, the user prepares files of the program core, microcode and input data. The software support system provides a documentation file, the load module, and a command file. The user executes the command file, which binds the files to the load module and places the user inside the execution module. The user must prepare the following:

- A description of the object machine in CALSIM.
- A program to run the object machine.

* A microprogram for the machine.

* One or more data input files for the machine.

* A file containing the auxiliary memories.

The environments provided by the compiler/simulator are:

TOP LEVEL -- Routes the user to the working environments.

COMPILER -- Reads, compiles, and lists the hardware description.

MICROFILE READ -- Reads microfile and checks for errors.

PROGRAMFILE READ -- Reads program file and checks for errors.

AUXMEMORY READ -- Reads auxiliary file and checks for errors.

TABLES -- Prints the tables produced during compilation.

LIBRARY -- Allows the library of descriptions to be examined.

DOCUMENTATION -- Allows the user to request helpful documentation.

SIMULATION DRIVER -- Allows the user to control the simulation.

SIMULATOR -- Performs the simulation entered from the driver.

Each environment has its own prompt character and its own set of commands. In each environment a "?" provides the user with a list of acceptable commands and "HELP" provides a brief explanation. During compilation the library file is built by the user from syntactically correct statements. Items removed from the library for use in a hardware description during compilation are also checked syntactically. The user may delete, search, and list library contents.

LANGUAGE AND SOFTWARE: THE COMPILER

The compiler was written using the syntatic analyzer portion of a skeleton compiler for LALR grammar as a guide. The skeleton compiler was written in XPL by the University of Toronto Group who prepared the Analyzer(2). The Lexical Scanner Portion was adapted from a compiler written in Cobol by David Levine, UTA. Since the language is a series of statements, error recovery is by statement. If the statement is acceptable, it is compiled; otherwise it is rejected as soon as the syntactical error is discovered, and the scanner moves to the beginning of the statement which follows the next period or reserved word signaling a new statement. No attempt is made to remove code already in tables. All of the hardware description data are converted into tables which will drive the simulator and be available to the user.

LANGUAGE AND SOFTWARE: SIMULATOR AND SIMULATOR DRIVER

Programming, now underway for the simulator and simulator driver, calls for the driver to pass control to the simulator and for the simulator to check for a return to driver after each clock pulse. The user will be able to set break points:

* On read to or write from memory.
* On read from or write to a given port.
* On the next n program steps.
* on the next n microprogram steps.
* On a certain clock settings.

Displays will be arranged so that they may occur:

* On direct command from the console.
* After n program or microprogram steps.
* At break only.
* At memory read or write.
* At port read or write.
* At chosen time settings.

The user can define 20 display configurations. A heading line is printed the first and each 8th time the menu is printed. A trace through the program or microprogram may be combined with display. The user may also reset the values in registers and on buses. The simulation is planned as interpretive execution of the logic introduced in the description of the components. Each component statement will have a series of condition/replacement clauses. A table which shows the timing sequence of the components will be used to control the operation order of the active components.

EXAMPLES OF THE CALSIM LANGUAGE

In the examples below, the user supplied identifiers and numbers are in lower case; CALSIM reserved words are in capitals; comments are enclosed by "/*" and "*/". CALSIM statements define the name of the system, logical components including memories, input/output, and connectors. Statements also set timing constants, send to or receive descriptions from the library, and wire the components together with a connect statement. Each system description in CALSIM begins with the system name followed by a series of statements describing the object hardware:

```
SYSTEM NAME trial-design.
...
/*the statements which describe the hard-
 ware goes here */

...
END OF trial-design.
```

The hierarchy statement will be used next to list each of the components of the system and their position in the hierarchy, from the top level starting with the system name, downward. All of the components must be included -- connectors, memories, peripherals, and logical units. Component names may have up to 30 characters; commas and spaces following commas are both optional:

```
LEVEL 1:
trail-design CONTAINS bigbus, unit1,
unit2, memory-unit, prt1, prt2.
```

The user further decomposes the machine description using the hierarchy statement. Up to 99 levels are permitted, allowing the hierarchy to decompose even a complex machine to the chip level.

```
LEVEL 2: bigbus CONTAINS sbus1, sbus2,
         sbus3, sbus4;
unit1 CONTAINS unit1-1, unit1-2, unit1-3;
unit2 CONTAINS unit2-1, unit2-2, unit2-3;
unit3 CONTAINS unit3-1, unit3-2.
```

Note that the language specifies that statements be ended by periods and clauses by semi-colons. Also in the example given, an error would be received for entry of "unit3" since that identifier had not been previously entered at level 1. The connector statement, used to represent wiring may begin with "BUS," "BACKPLANE," "CORD." All are equivalent to the compiler. Individual wires within connectors may be named (up to six characters) in addition to the required numbers. Individual wires may be tested in the logic statements using the names or numbers. Note that "IS" is optional in the statements below. Should there be too few or too many wire names, a warning will be given. The connector statement only defines the connector, the actual hook-up is made later with the "CONNECT" statement.

```
BACKPLANE: bplane IS NUMBERED (1-122).
BUS: sbus1 IS NUMBERED (1-12) NAMED d1,
d2, d3, d4, adr1, adr2, adr3, adr4, adr5,
ad46, cnt1, cnt2.
CORD: sbus2 NUMBERED (1-32).
```

The memory statement provides for five clauses to describe the memories, their logic, and connections. The memory statement may start with one of three reserved words -- MEMORY, MICROMEMORY, or AUX-MEMORY. The actual contents of these memories will be read into the compiler/simulator memory just before simulation and at the same time checked for proper format, width, and length. A memory statement follows.

```
MEMORY: memory-unit
SIZE = 8 * 1024;
PINS ARE NUMBERED (1-42) AND NAMED d1, d2,
d3, d4, d5, d6, d7, d8;
TIME IS 2:3:1:4:8;
IF PIN (6-9) = 7H AND PIN 14 = 1 THEN
MEMORY (PIN (10-24)) = PIN (d1 - d8).
/* the H is for hex.  Decimal, binary,
 and octal numbering systems may also be
 used.*/
```

The size clause shows the size of the memory in bits. The pin clause numbers and names the pins of the chip. The clock may have up to eight levels with each level containing one decimal digit. These values are saved in a table so that the simulator can determine the execution sequence of the logical components. If required, the time could have been entered more than once and a time test introduced into the Boolean expression. The peripheral and component statements are similar to memory statements and also use the time, pin, and variations of the If...THEN clauses. An example of the input/output statement follows.

```
I-0  terminal1
PINS ARE NUMBERED (1-32);
FORMAT IS ASCII;/*other I/0 formats
                 available */
CLOCK = 2:4:2:6:8;
TIE TO FILE 3;
IF PIN(1-4)=10H THEN PIN(5-12) = INPUT;
IF PINS(1-4)=11H THEN OUTPUT=PINS(5-12).
```

Replacement expressions support nine dyadic and seven monadic operators and also nested IF expressions. Examples of the nested IF and several of the operators follow.

```
CHIP: expno1 Pins are (1-44);
CLOCK = 1:4:5:3;CLOCK=2:3:1:4:8;
REGISTERS ARE a, f, h, l, sp, ix, iy;
SUBREGISTERS OF f ARE cry 1 zero 1 dm 6;
CASCADE b c INTO bc;
IF CLOCK = 2:3:1:4:8 AND PIN 4 = 1
THEN A = PINS (24-36) + B,
 IF a = 0 then z = 3;
IF PINS (36-40) = 14  then a = b:
IF PINS (36-40) = 1101B then a = RL a
```

The store and copy statements control the library use as shown below:

```
  STORE AS ureownname
  ...
  /* one complete CALSIM statement */
  ...
  END OF ureownname.
  COPY ureownname SUFFIX ab.
```

Finally components must be connected together using the buses and cords. This wiring operation provides the logical path for the signals to follow during simulation. The statement maps the wire numbers from the bus, backplane, or cord onto the pin numbers of the memories, I-0 devices, or other logical component.

```
CONNECT  bus3 TO chipl (1, 4, (7-12), 2,
         3)((1-12));
TO terminalone ((16-32)) ((4-12)).
```

CLASS EXPERIENCE WITH CALSIM -- COURSE DESCRIPTION

The language and the compiler/simulator were used and evaluated by a group of 17 graduate students in an introductory computer organization course at UTA. The students, all of whom had undergraduate degrees in fields other than computer science, had completed six to fifteen hours of graduate Computer Science prior to taking the course. By midsemester, students were expected to be familiar with the manufacturers literature, the DEC 20, and the compiler/simulator, and to know the text (20) chapters covering operations, number systems, Boolean algebra, gate networks, logical designs, and part of the operation of the Arithmetic-logic-unit (ALU).

For the mid-semester (take-home) examination, students received a microcomputer design with some fifteen explicitly wired components along with a detailed explanation of the action of each component. The students were required to design several of the units using gates, e.g., a unit was shown which could select the next micro-memory address from a ROM, the pipeline register, or the microprogram counter. The students had to design a selector switch that would place the correct value on the output pins of the unit based on the input from three control lines.

The term project, assigned during the second week of the semester, required the students to choose a microcomputer and to develop a block diagram using the components. They were then to describe in the CALSIM language the hardware represented by the block diagram and to process the description through the compiler. Finally, critiques of the CALSIM language, the compiler/simulator, and their use as teaching tools were required. To encourage searching for defects in the system, extra points were given for each Software Trouble Report. Several 30 minute lectures on the CALSIM grammar, language, and compiler/simulator were given during the third quarter of the semester. The students were permitted to work individually or in groups; a total of eleven reports were received.

CLASS EXPERIENCE WITH CALSIM - RESULTS AND CRITIQUES

Student program lengths varied from 95 to over 1000 lines of code. The components numbered from 12 to 20. The response to the language and grammar indicated that both were easy to learn and use. This was confirmed by the relatively long error-free programs.

BIBLIOGRAPHY

(1) Alexandrididis, N. A., "Bit-sliced Microprocessor Architecture", IEEE Computer, June 1978, pp. 56-80.
(2) LaLonde, W.R., An LALR Grammar Analyzer, University of Toronto, Computer System Research Group.
(3) Armstrong, J. R. and Woodruff, G. W., "Chip Level Simulation of Microprocessors", Computer, January, 1980, pp. 94-100.
(4) Barbacci, M. R., "A Comparison of Register Transfer Languages for Describing Computer and Digital Systems", IEEE Transactions, Vol. 24, No. 2, Feb. 1975, pp. 137-150.
(5) Bara, J. and Born, B., "A CDL Compiler for Designing and Simulating Digital Systems at the Register Level", Proceedings of the 1975 International Symposium on CHDL and their Applications, pp. 96-102.
(6) Borroione, D., "LASCAR: A Language for Simulation of Computer Architecture", Proceedings of the 1975 International Symposium on CHDL and their Applications, pp. 143-152.
(7) Bressy, Y., David, B., Fantino, Y., Mermit, J., "A Hardware Compiler for Interactive Realization of the Logical Systems Described in Cassandre", Proceedings of the 1975 International Symposium on CHDL and their Applications, pp. 62-68.
(8) Burris, H. R., "Instrumented Architectural Level Emulation Technology", National Computer Conference, 1977, pp. 937-746.

(9) Chu Yaohan, COMPUTER ORGANIZATION AND MICROPROGRAMMING, Prentice-Hall, 1972.
(10) Franta, W. R., and Giloia, W. K., "APL*DS: A Hardware Description Language for Design and Simulation", Proceedings of the 1975 International Symposium on Computer Hardware Description Languages and their Applications, pp. 45-52.
(11) Johnson, W. A., Crowley, J. J., and Roy, J. D., "Mixed Level Simulation from A Hierarchical CHDL, SIGDA Newsletter, Vol. 10, No. 1, January 1980, pp. 2-10.
(12) Kerridge, J. M., Willis, N., "A Simulator for Teaching Computer Architecture," SIGCSE Bulletin, Vol. 12, No. 2, July 1980, pp. 65-71.
(13) Lipovski, G. J., Hardware Description Languages: Voices from the Tower of Babel, Computer, June 1977, pp. 14-17.
(14) Lloyd, G. R., Van Dam, A., "Design Considerations for Microprogramming Languages", National Computer Conference, 1974, pp. 537-543.
(15) Marczynski, R. W., Pulczyn, W. T., Sochacki, J. M., "OSM Microprogrammed Hardware Structure Description Language", Proceedings of the 1975 International Symposium on CHDL and their Applications, pp. 172-178.
(16) Smith, D. R., "Computer Structure Language", Proceedings of the 1975 International Symposium on CHDL and their Applications, pp. 153-160.
(17) Stewart, J. H., "LOGAL: A CHDL for Logic Design and Synthesis of Computers", Computer, June 1977, pp. 18-26.
(18) Stine, L. R., Mowie, F. J., "A Position Paper on Extensions to the Computer Design Lanugage", Proceedings of the 1975 International Symposium on CHDL and their Applications, pp. 103-114.
(19) Su, Y. H., "Hardware Description Language Applications", Computer, June 1977, pp. 10-13.
(20) Bartee, T. C., DIGITAL COMPUTER FUNDAMENTALS, McGraw-Hill, fourth edition, 1977.

Opal - A Hardware-Independent Assembly Language for Education

Thomas G. Luce, PhD
Odessa College

Background

As computers become increasingly important in our society, concern is growing about developing computer literacy; that is, making people understand what computers are and how they operate.[1] The introductory course proposed in the Data Processing Management Association's model curriculum project suggests that computer literacy involves some understanding of computer hardware and processing concepts as well as problem-solving and programming methodologies.[2] Many of these topics can be rather abstract, and while our college students should be capable of logical, formal thought, I believe that a concrete, tangible approach often works best.[3,4]

A related problem in computer literacy is developing within the computing community itself. While I agree with Shelly and Cashman that an understanding of assembly language is essential for the professional programmer, many participants of recent DPMA curriculum model workshops would argue that this is not important for the business applications programmer/analyst.[5]

Both problems may be related to the diversity and complexity of currently available systems which obscure relatively simple, elementary concepts (the forest for the trees problem). If, as Abel argues, learning an assembly language makes it easier to learn other assembly languages as well as higher-level languages (and more efficient use of those languages), would we not then be well served by a simple "computer" and assembly language upon which basic concepts could be taught?[6]

Opal, the Odessa Pedagogical Assembly Language, is an assembler and simulator of a simple, hypothetical computer designed to be a solution to the problems above.

The Opal Computer

Opal is a binary, word-oriented computer with internal data representations in octal. Memory consists of 1000 sequential 36-bit words. The control unit consists of five 36-bit registers, a program counter, instruction register, accumulator, index register, and an extra (x) register. Figure 1 shows the Opal machine-instruction format. Each instruction has a 6-bit operation code (OP) followed by indirect (IB), index (XB), and immediate (UB) bits, and a 27-bit operand field.

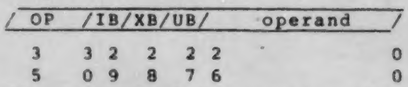


Figure 1. Opal Instruction Format

Opal supports absolute addressing along with indirect, indexed, and immediate addressing. The addressing logic allows indexed and indirect addressing to be used together. When indirect and immediate are specified jointly, an effective address is determined and used for the immediate operand. A combination of indexed and immediate addressing is not supported, and if specified, results in simple immediate addressing.

Subroutine calls and returns are implemented by means of a pushdown stack which is directly available to the user via push and pop instructions.

The instruction set of Opal (see Table 1), while limited, represents the major kinds of instructions currently available. These instructions include data moving instruction (LDA, STI, PSH, etc.), transformational instructions, arithmetic (IAD, ISI, MUL, etc.), logical (AND, COM, RCS, etc.), and branch instructions including unconditional (UNJ, RET, etc.) and conditional (IZJ, DSZ, AHJ, etc.) jumps.

Additional instructions provide for simplified input of decimal integers (RDI) and output of either decimal integers (PRI) or octal numbers (PRO). A final group of instructions (TON, DMP, DSK, etc.) have been included specifically to allow students to use traces and dumps.

Opal hardware recognizes a number of error conditions, including illegal memory references, unknown operation codes, division by zero, and stack over or underflow, among others. All errors are considered fatal and result in an aborted execution. During the abort, a dump is automatically generated of all registers and the simulated memory surrounding the instruction causing the error (see figure 2). Additional dumps list the contents of the subroutine stack and a jump history table detailing the jumps most recently executed prior to the abort.

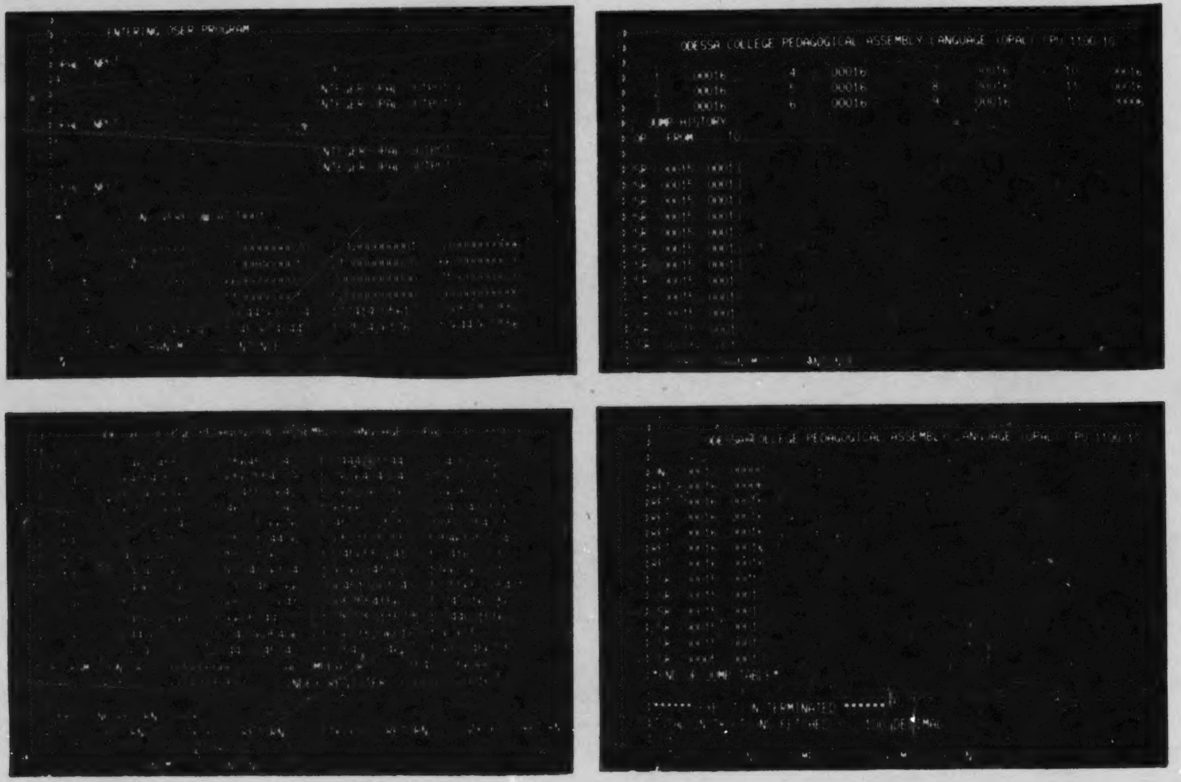


Figure 2. Execution listing for recursive factorial program including error termination dumps.

Opal maintains a count of instructions fetched, including indirect chains and aborts after 10000. An optional execution time switch allows this limit to be increased to 100000 for long programs.

The Opal Assembler

Opal uses a two-pass assembler that produces object code directly in simulated memory. The object program produced is executed immediately upon completion of the second pass, provided that no assembly errors have been encountered.

Each source line presented to the assembler has from one to four fields as indicated in Figure 3.

label OP [*] OPERAND $\left[\begin{Bmatrix} ,U \\ ,I \end{Bmatrix}\right]$ C

Figure 3. Opal Assembler Source Line Format

The label field is optional and provides symbolic address capabilities for the assembler.

Labels must begin in column one and may be up to ten characters long. Source lines with only a label are allowed. If any fields follow the label, they must be separated from the label by one or more blanks.

The OP field, beginning anywhere after column one, contains an operation code mnemonic or an assembler pseudo-op. An asterisk (*) immediately following an opcode indicates indexed addressing.

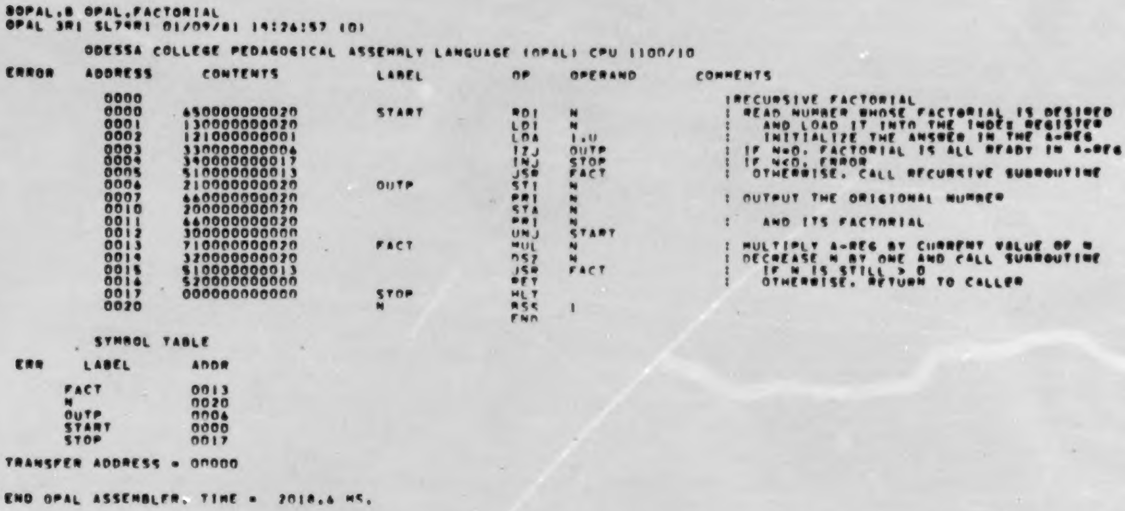
The operand field may contain a symbolic address, a constant, or a "$" representing the assembly time current location counter. Relative addressing is possible by adding or subtracting a decimal constant to a symbolic address or the current location counter, e.g., LDA TABLE+4 indicates that the accumulator is to be loaded from location TABLE plus 4.

The presence of a comma I (,I) immediately following the operand indicates indirect addressing. Comma U (,U) indicates an immediate operand.

The comment, or C, field begins with a semicolon Everything following the semicolon is ignored by the assembler. If three non-label fields occur on a source line, the third is automatically considered a comment line even if no semicolon is present.

The assembler supports four pseudo-ops that permit coding of decimal (DEC) and octal (OCT) constants, allocation of table space (BSS), and assembly termination (END). The END pseudo-op also allows the user to specify a transfer address.

Output from the assembler is a formatted source listing (see Figure 4) that includes the octal contents of all modified memory locations, a symbol table, and a listing of real CPU time used for the assembly. A table of error codes is added to the listing if errors were detected during assembly.



```
@OPAL,B OPAL,FACTORIAL
OPAL 3R1 SL79R1 01/09/81 19:26:57 (0)

        ODESSA COLLEGE PEDAGOGICAL ASSEMBLY LANGUAGE (OPAL) CPU 1100/10

ERROR   ADDRESS    CONTENTS          LABEL        OP    OPERAND      COMMENTS

          0000                                                        !RECURSIVE FACTORIAL
          0000    650000000020       START        RDI   N             ; READ NUMBER WHOSE FACTORIAL IS DESIRED
          0001    130000000020                    LDI   N             ;    AND LOAD IT INTO THE INDEX REGISTER
          0002    121000000001                    LDA   1,U           ;    INITIALIZE THE ANSWER IN THE A-REG
          0003    330000000006                    IZJ   OUTP          ; IF N=0, FACTORIAL IS ALL READY IN A-REG
          0004    340000000017                    INJ   STOP          ; IF N<0, ERROR
          0005    510000000013                    JSR   FACT          ;  OTHERWISE, CALL RECURSIVE SUBROUTINE
          0006    210000000020       OUTP         STI   N
          0007    660000000020                    PRI   N             ; OUTPUT THE ORIGIONAL NUMBER
          0010    200000000020                    STA   N
          0011    660000000020                    PRI   N             ;    AND ITS FACTORIAL
          0012    300000000000                    UNJ   START
          0013    710000000020       FACT         MUL   N             ; MULTIPLY A-REG BY CURRENT VALUE OF N
          0014    320000000020                    DSZ   N             ; DECREASE N BY ONE AND CALL SUBROUTINE
          0015    510000000013                    JSR   FACT          ;    IF N IS STILL > 0
          0016    520000000000                    RET                 ;    OTHERWISE, RETURN TO CALLER
          0017    000000000000       STOP         HLT
          0020                       N            BSS   1
                                                  END

        SYMBOL TABLE

ERR     LABEL       ADDR

        FACT        0013
        N           0020
        OUTP        0006
        START       0000
        STOP        0017

TRANSFER ADDRESS = 00000

END OPAL ASSEMBLER, TIME =  2018.6 MS.
```

Figure 4. Opal Assembler Source Listing

If no assembly errors were encountered, control is passed to the simulator which starts executing the object program at the transfer address or location zero if no transfer address was specified.

Because Opal is designed to be run interactively from a terminal as well as batch, special features are included to facilitate interactive use. When run from a terminal, Opal will send only enough information to the CRT to fill one screen at a time. Once the screen is full, the program waits for the user to press the transmit key before continuing execution.

When an input instruction (RDI) is encountered by the simulator, an input request is issued to the terminal user. Batch jobs do not receive this request, but the input is echoed to the line printer. Integer and octal output from the simulator is labeled for the benefit of the student.

Executing a halt instruction terminates the simulator which prints a line indicating the location of the halt instruction, milliseconds of real CPU time used, and the total number of instructions fetched. Error termination causes the same information to be listed in addition to the dumps discussed previously.

Debugging Aids

The dumps produced during an error termination, register and memory, subroutine stack, and jump history table, can be requested at any point in the program by including the proper OP in the source program. The Opal read command can be used for additional debugging. Read expects a decimal integer to be entered but will accept and perform DMP, DSK, and DJT. Demand mode dumps are formatted four words per line while batch dumps are written with eight words of memory per line. A read instruction will also accept TON, TOF, and HLT instructions. TON and TOF set a trace control flag, and HLT causes an immediate, normal program termination. After input of any valid debugging command, except of course HLT, Opal will again request the integer input. At this time an additional debug command or the required number may be entered. Input of a non-debug command or other invalid data will result in an error termination complete with dumps.

When the trace control flag is on (TON), information about each instruction executed is printed (see Figure 5) after the instruction has been executed. This information includes the instruction location and mnemonic as well as before and after contents of any registers or memory locations changed by the instruction.

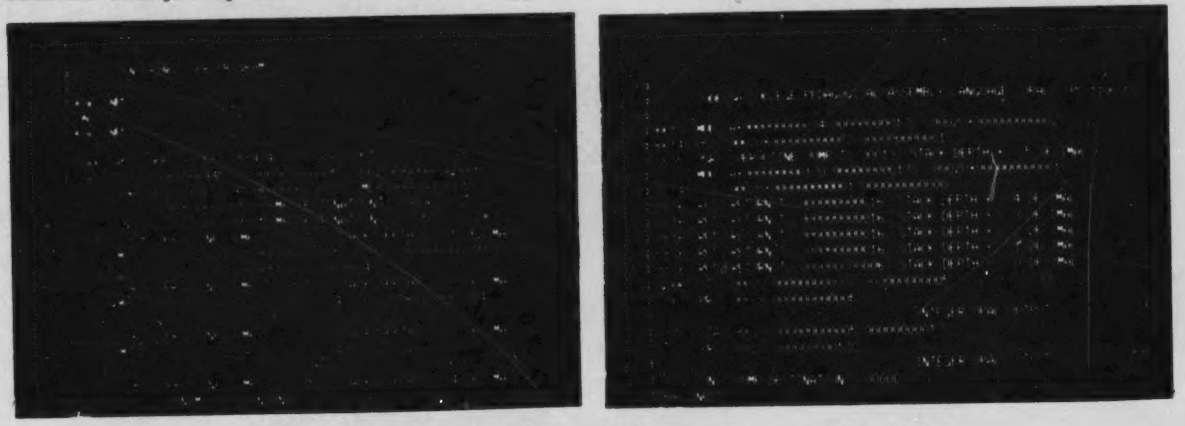


Figure 5. Recursive factorial program execution with input of TON activating trace mode.

The effective address and contents of locations used in memory references are displayed as are immediate operands when appropriate. Tracing conditional branch instructions displays the register being tested and the destination of the jump, whether taken or not.

Subroutine calls and returns list the destination of the jump and the depth of the subroutine stack after execution of the instruction. Push and pop instructions also cause the stack depth to be listed.

Execution of a TON instruction causes the simulator to enter trace mode and to remain there until the end of execution or until any TOF is executed. Selective use of TON and TOF allow the user to trace anything from single instructions to entire programs.

Current Use

Opal is presently being used at Odessa College in a freshman-level introduction to data processing for majors and non-majors. Two or three short programs required of students raise questions such as:

1) How much work can a computer accomplish in a single instruction?

2) How difficult is it for a higher-level language to compute A=B*D/E**5?

3) How can the computer tell the difference between data and instructions?

4) What happens if you try to execute data?

5) What happens if you forget a halt instruction?

6) Why are codes necessary for letters and symbols?

7) I want the computer to stop when the remainder is six; is it going to do that for me? (You mean I have to tell it to do that??)

8) How does the computer know where the data are anyway?

Most programs assigned use a subset of Opal, often only absolute and relative addressing along with simple branch and arithmetic functions. Questions, such as how do I make the computer do an integration?, are approached by demonstrating that complex problems can generally be broken down into a series of simpler functions. A lab requiring students to multiply and divide numbers using addition and subtraction may be assigned as a case in point.

After working with Opal in labs for approximately a month, we move to where we can now consider what the compiler/interpreter must be going through to do various requested functions. Opal is also used in lectures to demonstrate common CPU features. Addressing modes can now be discussed refering back to the simple addressing used in the lab. Indirect addressing can be compared with absolute addressing and memory modification. Indexed addressing can be related to base register addressing.

Opal has also been used for the first half of another class, Assembly Language Programming. In this course a heavy emphasis is placed on the relationship between machine architecture and machine instruction formats. Students use Opal to an assembl source line into machine code and back to source. Addressing modes are covered in detail with special consideration given to flow diagrams depicting effective address calculation.

After a brief study of the Opal instruction set, assembly programming concepts are presented and demonstrated with Opal programs. Topics include loop control and table handling techniques for reentrant and non-reentrant applications.

Subroutines are discussed using built-in Opal features and several other commonly used procedures. Five or six methods for argument passing are considered to prepare the student for various approaches encountered in real systems.

Finally, recursion is introduced by Opal's built-in subroutine stacks and user-defined stacks. From this framework, problems related to recursion in higher-level languages are touched upon briefly.

After seven or eight weeks of Opal, the study is shifted to the UNIVAC 1100/10. Concepts introduced with Opal are now applied to a real machine with multiple and overlapping registers, varied addressing modes, partial word capabilities, and a larger instruction set.

Concepts of looping and table handling introduced with Opal are quickly carried over and expanded. Assembly subroutines are written to be called from Fortran of Cobol main programs. Subroutines linkage and argument passing, first discussed with Opal, relate directly to these real life situations.

While no formal evaluation of Opal has yet been attempted, instructors using the language have an intuitive feeling, based partly on student questions and exam results, that it fulfills a purpose in our educational program. This belief is further supported by students working on IBM System/370-type equipment who report that Opal has helped them understand those machines.

Technical Considerations, Transportability and Other Loose Ends

Opal is currently implemented on a UNIVAC 1100/10 operating under the 1100 Executive (36R1). Most of the code is in Fortran 77 (FTN 9R1), but some sections are in 1100 assembly. The most important assembly code deals with implementation of Opal's shift instructions. Less important assembly code determines CPU time and operation mode (batch/demand).

Conversion or elimination of assembly code should present few problems for transportability. More significant problems may be encountered by sites which do not yet have Fortran 77, as Opal makes extensive use of block IFs and to a lesser extent, other features not found in standard Fortran IV.

We are attempting to maintain two equivalent versions of Opal at Odessa College. One is a standard version which should be transportable within the limitations mentioned above. The second version is optimized for the 1100's operating system. This version uses the system source input routines (SIR$) and behaves like a standard system language processor.

Both versions make limited use of standard processor call options to turn off the demand mode flag and to override the smaller of the instruction fetch cycle limits. Both functions could easily be changed to use sense switches or some other appropriate feature.

Currently we plan to add several features to Opal. One change would permit read instructions to detect and branch on an end-file condition. A more important planned change will allow text I/O. This will not only allow beginning students to prompt for input and label output, but will also allow advanced students to study real life I/O formatting problems.

We believe that an elementary understanding of internal computer operation is not only an important part of computer literacy, but that it also facilitates understanding of other computing concepts and languages. We also believe that the complexity of modern computers can overwhelm the novice programmer. Opal was developed in response to these needs and limitations. While no formal evaluation has yet been attempted, progress is being made toward our goals. We invite others who share our concerns to investigate Opal and to use it if appropriate.

REFERENCES

1. Gensler, Philip. President's message, The Texas AEDA Newsletter, Dec 80.

2. DPMA's Model Curriculum Project, revised Oct 80.

3. Flavell, J.H. The Developmental Psychology of Jean Piaget, Van Nostrand Reinhold Co., 1963, esp chap 6.

4. Inhelder, Barbel and Piaget, Jean. The Growth of Logical Thinking from Childhood to Adolescence, Basic Books, 1958.

5. Cashman, Thomas J. and Shelly, Gary B. Introduction to Computer Programming IBM System/360 Assembly Language, Anaheim Publishing Co., 1969.

6. Abel, Peter. Programming Assembler Language, Reston, 1979.

TABLE 1
THE OPAL INSTRUCTION SET

mnemonic	octal	description
AND	40	logical product of [A-reg]* and [EA]
ANJ	37	jump to EA if [a-reg] is negative
AZJ	36	jump to EA if [A-reg] zero
COM	50	1's complement of [A-reg]
DIV	72	[A-reg] divided by [EA], quotient to A-reg, remainder to X-reg
DJT	05	dump jump history table
DMP	03	dump memory from word zero thru EA
DSK	04	dump subroutine stack
DSZ	32	decrement [EA] and skip next instruction if zero
HLT	00	stop simulation
IAD	14	integer add [EA] to [A-reg]
IAI	16	integer add [EA] to [I-reg]
INJ	34	jump to EA if [I-reg] is negative
ISB	15	integer subtract [EA] from [A-reg]
ISI	17	integer subtract [EA] from [I-reg]
ISZ	31	increment [EA] and skip next instruction if zero
IZJ	33	jump of EA if [I-reg] zero
JSR	51	jump to subroutine at EA
LCS	43	left circular shift of [A-reg]
LDA	12	load A-reg with [EA]
LDI	13	load I-reg with [EA]
LLS	44	left logical shift [A-reg]
MUL	71	multiply [A-reg] by [EA]
NOP	77	no operation
OR	41	logical sum of [A-reg] and [EA]
POP	53	remove top entry from stack and place in A-reg
PRI	66	print [EA] as an integer
PRO	67	print [EA] as 12 digit octal
PSH	54	push [EA] on subroutine stack
RAS	47	right arithmetic shift of [A-reg]
RCS	45	right circular shift of [A-reg]
RDI	65	read an integer and store at EA
RET	52	jump to address on top of subroutine stack
RLS	46	right logical shift of [A-reg]
STA	20	store [A-reg] at EA
STI	21	store [I-reg] at EA
STX	22	store [X-reg] at EA
TOF	02	turn trace mode off
TON	01	turn trace mode on
UNJ	30	unconditional jump to EA
XIA	11	exchange [A-reg] with [I-reg]
XOR	42	logical difference of [A-reg] and [EA]

* [] = contents of, A-reg = accumulator, EA - effective address
I-reg = index register, X-reg = extra register

PRE-ASSEMBLER:
A BRIDGE [COURSE]
OVER TROUBLED WATERS

Dr. Ronald C. Turner
Roland J. Keefer

Department of Mathematics
and Computer Science
Eastern Washington University
Cheney, Washington 99004

Although some may view the bridge course as an outrageous extravagance, we at Eastern Washington University are convinced that Computer Science 298, Pre-Assembler, is a luxury well worth the cost. The course is not required for computer science majors, and it is not an official prerequisite for the course, Assembly Language Programming. In fact, it is not even listed in the official university catalog, but relies on word of mouth publicity. Yet, enrollment has grown from 4-5 to about 25 since it was first offered two and one-half years ago. Although hard follow-up data are not available, instructors for the assembler course note that graduates of Pre-Assembler invariably enter with increased maturity, a greater understanding of fundamental concepts of machine language and architecture, and with a more positive and expectant attitude, compared with those who enter with only the required working knowledge of a computer language. Grades have increased by approximately 1 unit. One instructor, not knowing which students had taken Pre-Assembler, pinpointed 13 out of 15 people in the given section. This was due primarily to enhanced insight and understanding. This observation is not very surprising, nor is it terribly interesting, from a curricula planning point of view. On the other hand, we feel that Pre-Assembler is unique and interesting for its own content and for its systematic treatment of the pitfalls awaiting the neophyte assembly language programmer.

Every legitimate topic in the curriculum could be accompanied by a list of identifiable inhibitors. These are commonly recognized problem areas which can easily cause a student to stumble and which require particular effort and care by the instructor. Inhibitors are not necessarily spawned by the difficulty of the content itself. Instead, they are often generated by a quantum leap between a student's prior course work and the content of the current course. Assembly language programming appears to be a prime example of material which, though not intrinsically demanding, packs an unusually high concentration of intellectual trauma for the uninitiated.

Here are some characteristics of assembly language programming which the typical undergraduate confronts for the first time:

1) A totally alien instruction set: Even with the required Fortran, Pascal, or Basic, the student has at least had whole, meaningful words in the instruction sets. Now, there are only cryptic mnemonics presented in some strange vertical order. They do not appear to resemble the commands of high-level languages in any way. There is no equal sign, at least not for assignment, as in Basic or Fortran. And even something as fundamental as IF is nowhere to be found. Unnerving indeed!

2) Miniscule increments of task completion per instruction: Once the student has learned enough commands to perform useful work, he or she is overwhelmed initially by the high ratio of instructions per actual task unit: a dozen instructions just to move a field of alphanumeric characters into an output area. And we thought Cobol was verbose!

3) A nearly total lack of semantics: The student fresh out of Fortran will have exerted considerable effort to master the intricacies of calling conventions and parameter passing, globality and variable protection, and perhaps some basic control structures. But with an assembler, none of these come with the package. Even such a burning issue as whether recursion is possible or not becomes a nonissue. The programmer can do virtually anything he or she desires. This instant omnipotence

becomes instant impotence, leaving many in a nervous state, bereft of their comfortable list of "shalts" and "shalt nots."

4) Lack of data structures in the language: The lower division student has just become proficient in translating interactive algorithms into loops within the code and into arrays for storing and accessing data in some orderly manner. Now the very mechanics of allocating, manipulating, and accessing arrays are dumped back into his or her lap. Even finding A(I) becomes a major task!

5) Forcible presence of the architecture of the machine: Computer hardware is not the cup of tea of the typical undergraduate programming language student. And yet, here it is, explicit, detailed, and essential. One must know precisely the system of registers used in the system. How wide are they? Are they general purpose...really? How does one move things on and off the stack? Which instructions apply only to the stack and stack pointer? Are there separate stacks for the system and for the user? How is memory laid out? Where are the critical and protected areas of memory, portions which the user should avoid? What is the word length of the system? How does the programmer handle numbers which exceed this limitation? And worst of all, how many ways are there for the user to address memory? In most cases, not a single one of these topics was even mentioned in the prerequisite programming courses. Depending on the particular student's inclinations, this confrontation with the system can be overpowering.

6) Need for awareness of speed vs. space tradeoffs in code: A typical student completed Fortran by getting some minimal set of programs just to run! The difference between a program which runs and one which does not is a far cry from the difference between register-to-register and memory-to-memory accesses within a critical loop. Yet the assembler student is now forced to consider optimization of this latter variety, pondering the consequences of wasted nanoseconds.

7) Nondecimal base arithmetic: It is bad enough that the sequence of instructions, in mnemonic form, appear cryptic. But the object code itself is in octal or hexadecimal, making debugging at this level laborious.

8) Operator-operand format: Every operation, no matter what the particular application, is forced into the mold of an instruction having a single operation followed by zero, one, or two operands. Compared to any high-level language, this strikes the novice as a terribly stifling requirement for a language.

9) Testing and branching based on condition codes or flags: Life was rather uncomplicated when an algorithm, no matter how complex, could be reduced to a series of IF statements, perhaps nested. (Of course, this reduction itself took considerable effort.) But now, there is no IF, and we must resort to esoteric tests of bits in status registers. Then we must branch or jump accordingly, making certain that we do not invert our intentions in the process. And as we branch, we may have to watch continually for the interval of the code we branch over. If, as on several smaller machines, the displacement would be too far, then another instruction (JUMP, perhaps) and additional memory must be used.

10) On-line debugging techniques: Previous to this, the student's bag of tricks for debugging consisted mainly of STOPS (in Basic) and of extra WRITE statements (in Fortran) to follow the execution of a program. Now the programmer is given the opportunity to single-step through machine-level code, examining and changing contents of memory and registers. This is a whole new skill, comparable to learning the subset of another programming language. It is particularly disconcerting to many that they never learned to efficiently debug a program in a high-level language. Now, they find themselves forced to expand upon inadequate skills.

11) Aura and mystique surrounding the assembler language "priests": Irrational or not, a student who has learned the difference between systems and application programmers and between levels of languages probably has learned to respect, even venerate an assembly language programmer. This fear of the ultimate can generate an unhealthy inhibition in the would-be assembler student.

12) Unreadability of code: Even when commented and structured, assembler code is tough to read. And since students learn quickly by reading examples from their texts and modeling their own programs after those, their best source of help now turns into yet another barrier which is unfortunate.

13) Lack of helpfulness of manuals: Among the scores of Basic and Fortran textbooks, the student can normally find help at a level which is appropriate to his or her needs. The typical

undergraduate rarely has to consult a system manual at all. But for a problem in assembler, he or she will most likely be driven to seek help in the vendor's documentation. This is often an exasperating experience for a good programmer. For a beginning student, it may be fatal.

14) Extra system layers: The high-level language user generally debugs programs from within the language itself. The assembly language user must learn a new level of system interaction: the system utilities. Some individuals never learn to distinguish between the utility program, the assembler, and their own program! Furthermore, the vagaries of systems programs constitute an imposition on the already too short academic term.

Officially, we as a profession continue to require knowledge of an assembler to achieve an:

1) understanding of machine-level execution.

2) appreciation for issues of optimization of high-level code.

3) ability to link to existing assembler routines.

4) ability to write original code for time-critical routines.

5) ability to encode entire programs in assembler.

Although reason five is a weak justification for learning an assembler, a particular application may require an extremely close grasp of data or forms layout. Also, if a program is to be compiled very frequently, then an assembler would speed up that portion of development time. And of course, single-step debugging in assembler code, particularly for small systems, can yield bug-free code in a shorter time.

At a deeper level, we hope that mastering an assembler will endow the student with that elusive virtue called maturity. We claim that this sort of maturity is gained primarily by meeting and overcoming the inhibitors listed above. The discipline of learning to use a specific instruction set for writing code is quite secondary, perhaps even incidental.

Admittedly, no single item from our list is terribly traumatic. The trauma for a student is synergistic; it consists in having to become sensitized immediately to a wide spectrum of concepts. Everything in our list must be faced squarely before a single program of any consequence can be written and debugged.

The rest of the paper will describe the Pre-Assembler course. The importance of the paper will be in the course's point-by-point antidote to the 14 ailments.

Pre-assembler was born out of the recognition of clearly identifiable deficiencies/needs within the curriculum itself. Since it does not treat a single body of knowledge, there is no single textbook which could satisfy its unique requirements. Instead, the course is really three contiguous units of material, each with its contribution toward eliminating assembler trauma.

The first three-eighths of the course is based on the use of the HP33E programmable calculator. All of the students have had programming experience, mostly in Basic. So, they are accustomed to the process of reducing problems to algorithms, then to code. But they soon experience the flavor of an assembler-level world when they find themselves forced to consider programs in units of miniscule steps, rather than in the sentence-level instructions they had used in a higher-level language. Each instruction has from zero to two operands, similar to the format of assembler codes. Data may be carried within the program itself, as in assembler, but they are always in a sort of immediate mode. They quickly get comfortable with seeing their logic represented as numbers, since the calculator stores programs as sequences of location codes for the various keys. These numbers are the sole identifiers of the instructions, as the programmer examines a program already in memory.

The architecture of the HP calculator is well-suited to teaching assembler concepts. Storing and recalling data from the seven registers, a feature of the calculator, is a pattern for the most frequent type of programming in an assembler. The students become conscious of the stacking features of the HP series, and this, in turn, leads easily to the notion of a stack pointer.

Testing and branching with a calculator is a combined process which is remarkably similar to an assembler's methods. The user learns that the equalities and inequalities are really logical assertions which generate Boolean truth values. From this, it is relatively trivial to speak later of logical test instructions in assembler, which set and clear the various status flags. In fact, once students have learned how the calculator hardware forces a branch around the next instruction if a value is false, they are relieved to discover the increased flexibility of the assembler.

Program debugging for most beginning students is a hit-or-miss affair. Except for the strategy of keying in a whole program, and then simply hoping that it works, the only methodical approach to debugging with the calculator is through single-stepping. We discover several benefits for the purposes of the course. First, this is the most universal means available on interactive computing systems today. Then, the process reinforces the notion of code as numbers; the student reads his program in its numeric representation, just as he will read octal or hexadecimal code while debugging an assembler module. And while single-stepping through calculator code, the student examines and changes registers, alters code, and forces execution to restart at any point in the program; the entire process is analagous to the techniques used with most interactive debugging utilities.

The students work on programming problems which force them to exploit the instruction set of the calculator. The programs also require some ingenuity in order to fit within the constraint of the 49-step limit of the machine. To nurture a respect for software, of whatever dimension, each assignment must be submitted with proper documentation and full user instructions. If the instructor cannot make the program run solely on the basis of the student's written instructions, then the assignment is returned for revision.

The second module of the course, about six class sessions, is a brief treatment of number systems. We review place value notation in decimal mode, then discuss the general case using algebra. In this way, we attempt to avoid base conversions from becoming purely mechanical. The students then perform several conversions among decimal, hexadecimal, octal, and binary systems.

During this unit it is natural to introduce the topic of two's-complement arithmetic. Since we are not tied to a particular computer at this point, it is easy (and entertaining!) to discuss the motivation of complement notation inductively and to use a hypothetical machine. When we pretend that our computer cannot subtract, only add, then we are forced to invent artifacts like sign and carry bits and to watch for phenomena like over- and underflow. In particular, we demonstrate that complement arithmetic is possible in any number system. Using base 10 with a very restricted register width on our hypothetical decimal computer (having ten-state flip-flops) provides a light-hearted and painless transition to this otherwise nasty topic.

For the remaining four weeks of the course, Pre-Assembler becomes assembler by induction. We use the 6502 microprocessor which is imbedded in the Apple II computers in our student laboratory. The students receive no instruction set summary as yet, nor are they told how to use the line-by-line assembler provided by Apple. But now that they move easily in a world of machine-level instructions, they know quite well the sorts of operations which a microcomputer should accomplish. So we proceed inductively to discover a subset of the 6502 instruction set. We give them only the numeric op codes, which they laboriously enter in main memory one byte at a time. But even this is enough to allow them to write and execute programs and to single-step through their code.

Within two weeks of the end of the course, we distribute the full instruction set for the processor, together with mnemonics. Then we show them how to use the mini-assembler. Since they have been working in a state of language starvation for weeks, the embryonic assembler looks as powerful to them as a PL/1 or Pascal! Our assembler is a line-by-line interpretive assembler and hence admits of no labels. It requires the user to hand calculate all relative branching instructions, an extremely useful exercise in relative addressing, hexadecimal arithmetic, and two's-complement notation.

One capstone experience for the course is for students to master how an eight-bit processor can work with large numbers. This requires a full understanding of how the computer handles signed arithmetic, and how the various status flags are used by the application programmer. We find that the notions of both overflow and carry as well as underflow and borrow are well mastered through this exercise.

Almost at the end we address memory only in absolute and immediate modes. The 6502, having eight addressing modes with two index registers, provides as rich as repertoire as the student shall encounter anywhere. We survey the additional modes briefly at the end of the course.

As with all bridge courses, the real rewards materialize in the courses following. And indeed the level of expectancy and overall understanding among pre-assembler graduates in our UNIVAC assembler course is dramatic. It is refreshing to offer a course which was first developed from a philosophical/pedagogical standpoint. After the philosophy was analyzed, the mechanistic portion was then specified. Often, courses evolve in reverse order--the actual pedagogy is an attempt to rationalize hardware dictates. We feel that the design and content of Pre-Assembler have earned it the rights of a legitimate offering for years to come.

COMPUTER ASSISTANCE IN THE GERMAN INDIVIDUALIZED INSTRUCTION PROGRAM AT THE OHIO STATE UNIVERSITY

Heimy F. Taylor

Department of German

In January of 1976 the College of Humanities at the Ohio State University (OSU) was awarded a major grant from the National Endowment for the Humanities (NEH) to develop and implement individualized instruction programs for Arabic, French, German, Latin, Russian, and Spanish. This enormous undertaking was accomplished in several stages. All languages, except German, conducted their pilot programs for the first two elementary courses from September 1977 through July 1978 and were fully operational by the beginning of the academic year of 1978/79. At that time, two intermediate courses also became available. The development of the German program required more time because Professor Werner Haas, the coordinator, decided to create all materials, including a computer-assisted instruction segment, rather than adopting existing ones.[1]

The Department of German was the only one to incorporate a computer program with texts, tapes, and films. Professors Werner Haas and Heimy Taylor had already designed a computer program for traditional classroom sections. TUCO, Tutorial Computer, is a comprehensive tutorial program that includes all the basic German grammar. It is offered to German students free of charge and on a voluntary basis. TUCO has been in use since 1973 and can be considered one of the first comprehensive computer programs in German, if not in any foreign language.

Foreign languages have had quite a struggle with computer programs. In the late sixties, foreign language enrollment and competency levels began to decrease gradually but steadily, especially when one university after another dropped foreign language requirements for graduation. In the early seventies foreign language educators had to develop new programs or revise old ones. Teachers reacted in various ways to the crisis. Two of the numerous remedies attempted to booster enrollment were: the implementation of individualized instruction programs and the use of computers. Developing computer programs to teach foreign languages was attempted at several institutions in the early seventies.[2] Initially, results were quite disappointing; foreign language teachers themselves were reluctant to work with computers. They felt threatened in a world being taken over by technology. At the same time, expenses precluded developing programs, even if foreign language teachers were positively inclined towards the idea. In addition, the quality of some of the first programs left much to be desired, especially when they did not offer more than what a good programmed text could have done.[3] Critics were only too eager to question whether CAI could be effectively used in foreign language teaching. However, over the last two or three years, computer programs in foreign languages seem to have mushroomed all over the country. Solveig Olsen, in an interesting and highly informative article, "Foreign Language Departments and Computer-Assisted Instruction: A Survey," stated, "It appears, then, that the computer, and especially the microcomputer, is about to become accepted as a useful aid in foreign language instruction."[4] The article lists sixty-two foreign language departments with CAI programs, representing fifty-two institutions in twenty-four states and Washington D.C.[5] While this list certainly is encouraging, and shows a marked increase over the last three years, remember that the list of institutions not using computer programs in foreign languages is much longer. Yet, another positive change can be observed. Almost all major professional meetings of foreign language educators, such as the Northeast Conference on the Teaching of Foreign Languages, the annual meeting of the American Council of Teachers of Foreign Languages, the Modern Language Association meeting, and others devote at least one session to computer-assisted instruction programs, or the use of computers in foreign languages and literature in general. At the recent meeting of the Association of American Teachers of German in Boston (November 1980) a session entitled "The Role of the Computer in Foreign Language Instruction" was offered and very well attended. Even foreign language teachers seem to have finally accepted computers in their profession. Not quite; for every supporter of CAI programs, there are at least a dozen opponents or skeptics. This is partially due to the fact that the effectiveness of CAI programs in foreign languages never has been proven or established. What the profession really needs at this time are comprehensive performance studies which show whether or not computer programs actually are effective. The Department of German at OSU has conducted pilot studies, but results were inconclusive; more elaborate studies are planned for the near future. However, OSU has established that students like computer programs and consider them private tutors. An attitude questionnaire conducted during the academic year of 1974/75 and in the spring of 1976 indicated a high level of satisfaction among students, particularly those who used TUCO regularly.[6] Since then spot checks have had the same results.

This factor, plus a personal conviction regarding the usefulness of CAI programs in foreign languages played a major role in the decision to make CAI an integral part of our individualized German program. The natural relationship between individualized instruction and computer-assisted instruction is obvious when one examines the special features which characterize both: 1) variable pacing: students set their own pace of learning rather than follow the goals set by the department or an instructor; 2) one-to-one tutoring; 3) options on the selection of material at a given time; 4) eliminating unsatisfactory performance level (fixed standards of competence); and 5) branching options according to performance level. Olsen also points out, "The computer's effectiveness in assisting self-paced learning has resulted in numerous programs completely independent of any course format or textbook. Such materials are praised for being well suited for those who, for example, only need to brush up a little to pass a proficiency examination."[7]

As mentioned before, the Department of German at OSU has had a computer program for the elementary courses since 1973. This program was developed independent of any textbook. It is therefore possible to change textbooks without major revisions of the computer program.[8] When we developed our individualized instruction program, we felt that the total package, texts, tapes and CAI program, should be coordinated. Therefore, the CAI program which we developed for individualized instruction (I.I.) contains the same vocabulary and grammatical structures. Generally speaking, however, the two computer programs DECU, Deutscher Computerunterricht, for I.I. and TUCO, Tutorial Computer, for our regular classroom sections, are very similar. Both were written in the IBM Coursewriter III language and provide students with individualized tutorial instruction for German grammar. They use principles and practices of the tutorial methods, and enhance and fortify a program of well-balanced language instruction. The tutoring takes place on a one-to-one basis, and all instruction is geared to students' individual responses. The computer's ability to store a large quantity of information related to a student's most common mistakes in learning German provides tutorial advice which helps students overcome unchecked errors and reinforce grammatical patterns and structures. They respond to advice and hints normally given by an experienced teacher.

While TUCO is an adjunct to our classroom sections, DECU is fully integrated into the German I.I. program and is a required part of students' work. Their work is checked as printouts are brought to the Learning Center. CAI work is not graded, but students receive advice according to the analysis of their grammar practice on the computer terminal.

Specifically, how does the program work? At the beginning of each quarter students attend an orientation meeting where the I.I. program is explained; materials are introduced, the Learning Center is shown, and the computer program is demonstrated. From then on students are on their own. After they have studied a particular grammar segment, they are instructed in their text to do the computer exercises. Students must sign on with their individual number, and after they have selected a grammar segment, a brief text is typed out explaining the main features of the particular grammar topic. Then students must do exercises consisting of fill-in-the blank, sentence completion, synthetic exercises, or translations. If an answer is correct, they are branched to the next problem. In the case of mistakes, tutorial help is given regarding the nature of the mistake. Of course typing or spelling errors could not be programmed, and if students make typing or spelling mistakes, they are informed that their answer is wrong and that they should try again. After completing a segment, students are given a score.

The program distinguishes between a good and poor performance. If students do very well, they can skip a number of exercises in a given segment; if they performed poorly, they are asked to do the section over again.

What are some of the results of the German I.I. program? First, the program has contributed to a small, but steady enrollment increase, even though I.I. students earn on the average fewer credit hours per quarter than those taking the classroom sections, approximately three vs. five. Features of I.I. are the opportunity to work at ones own pace, enabling students to take two or three credit hours of German in addition to a regular course load; or to go slower because other classes are demanding most of their time in a given quarter; or to work longer in order to acquire the necessary knowledge, because quite a few are not motivated enough to work regularly and steadily on their own in order to earn as many credits as they would in a classroom. Over fifty percent of those students enrolled in the I.I. program indicated that they would not have been able to study German if I.I. had not been available.

A pleasant surprise has been the high performance level by I.I. students in unit tests. The testing in I.I. and in classroom sections is totally separated. To receive credit, a student must pass all tests with eighty percent or better accuracy. Although we have three versions for each unit test, only eight students had to take a test three times, and about five percent have had to take a unit test twice. The German Department has the policy that students must accept the grade they receive, as long as it is above eighty percent. During autumn, winter, and spring quarters of 1978/79 the average grade of 173 students who had no previous German was 91.5 percent. For 144 students who had previous German, it was 91.1 percent. This trend has continued in subsequent quarters. The high test results are particularly surprising considering average scores in other departments have been lower, and more students have taken their tests more than once. Perhaps our tests are easy and we grade leniently? This question has been asked, of course. Yet outside evaluators from NEH and other institutions found our program quite demanding, and colleagues in other language departments have commented they consider our program excellent, and that we simply prepare students well before they take their unit tests. We would like to submit, however, the opinion that the computer program which we incorporated has made a difference. Since all our students should complete CAI exercises before they take a test, we know that most practice their grammar regularly. The general trend shows that students who had German instruction prior to enrolling in an I.I. course practice fewer hours with the computer than do those who had no prior German experience. The result, as was cited above, is that the average grade of those students with no prior German is about the same as the one of those who had German before.

We have not been able to establish any correlation between time spent with the computer and grades and number of credit hours earned. We have discovered, however, that each quarter a number of students visit the Learning Center only when they want to take a test. This means they did not have computer printouts checked, and our records indicate, these same students generally did not use the computer at all. They do well, probably because they had several semesters of German in high school and were able to work through their materials in relatively short time. They probably could have passed a proficiency examination if they had chosen to do so. Apparently, they enrolled in I.I, in order to earn credit hours with a good grade. Over the last three quarters, winter, spring, and autumn quarters of 1980, instructors in the Learning Center did not ask for computer printouts. We wanted to see how many students would not use the computer. Of the 287 students enrolled in I.I., 43 earned an average of 2.2 credit hours with an average grade of 3.77, or almost A-, without using the computer program at all. The other students used the computer program approximately 650 hours in those three quarters. We did not detect a pattern: in fact student usage ranged from less than sixty minutes to over fifteen hours per quarter. This range had something to do with the fact that instructors did not consistently ask to see printouts.

Another factor should be mentioned here, namely, how did I.I. students fare when they went into the classroom after they had completed beginning German in I.I? At the present time we do not offer any other German course using I.I., therefore students wishing to continue with their German instruction must enroll in a regular classroom, particularly those students who have to fulfill a four quarter language requirement. So far we have found that almost all of our I.I. students have been able to continue in classroom sections if they chose to do so. In the beginning their oral skills are usually inferior;

however, they are particularly strong in their knowledge of grammar and score high on tests requiring it. Is this a result of our computer program in I.I.? We are inclined to think so. Classroom teachers continuously comment that I.I. students have little or no difficulty in making the transition to the classroom. Students have the option of enrolling in a basic (four-skills) course, a conversation, reading in the humanities, or reading in the sciences track. Those who feel especially weak in their oral skills generally enroll in one of the reading sections, a fact which applies to the classroom students also. Most I.I. students register for the four-skills track. Regardless which track I.I. students choose, their grades indicate that their transition was successful. Of the 43 students who have continued in 103 ten received an A, elven an A-, three a B+, four a B, four a B-, three a C+, three a C, one a C-, one a D+, three an E. It should be noted here that most 103 courses are taught by faculty while all I.I. courses are taught by Graduate Teaching Associates. In a recent questionnaire (autumn 1980 and winter 1981) we asked I.I. students if they felt that I.I. had prepared them well enough for the classroom? All of them answered in the affirmative.

How do I.I. students themselves feel about CAI? Of the 500 plus students who have participated in I.I. since its beginning, about 36 percent considered the program "very valuable," 58 percent "useful," and 6 percent "not useful." Occasionally students comment that they "consider the CAI part of I.I. the best."

In our experience with computer-assisted instruction in German, we consider DECU a very valuable segment. We feel that this program is of great help, especially to the weaker students; they become aware of their deficiencies as clues lead them to correct answers. CAI provides an opportunity to practice and review over and over again, a feature so important for weak students. In addition, CAI teaches not only grammar, it requires attention, concentration, and self-discipline which are necessary for developing successful study habits.

As far as general German language instruction is concerned, we feel the computer should assist in the total teaching effort. We view our programs not as an autonomous teaching device divorced from the teacher, the classroom, or the Learning Center in I.I., but rather as a valuable addition which can enhance and fortify a program of well balanced language instruction.

Notes:

1. For a detailed report on the four year experiment in Individualized Instruction in six foreign languages at the Ohio State University, see Leon I. Twarog and E. Garrison Walters, "Mastery-Based, Self-Paced Instruction in Foreign Languages at Ohio State University," *The Modern Language Journal*, 65 (1981).

2. Some of the institutions which experimented with CAI programs in foreign languages are: Stony Brook, Stanford, Dartmouth, the University of Minnesota, M.I.T., the University of Illinois, the University of Southern California, and others.

3. G. E. Nelson, Jean Renard Ward, Samuel H. Desch, and Roy Kaplow, "Two New Strategies for Computer-Assisted Language Instruction (CALI)," *Foreign Language Annals*, 9 (1976).

4. Solveig Olsen, "Foreign Language Departments and Computer-Assisted Instruction: A Survey," *The Modern Language Journal*, 64 (1980), p. 345.

5. Olsen, p. 341.

6. For detailed results see Heimy Taylor, "DECU/TUCO: A Tutorial Approach to Elementary German Instruction," *Foreign Language Annals*, 12 (1979), pp. 289-291.

7. Olsen, p. 344.

8. Since the development of TUCO in 1973 we have indeed changed textbooks once and adopted a second edition of our present text.

COMPUTER-ASSISTED INSTRUCTION IN MELODIC COMPOSITION

Paul E. Dworak
North Texas State University
Denton, Texas 76203
(817) 788-2791

The art of composing music involves creative thought. However, if every composer's style were completely unique, no one could comprehend music. Composers create patterns of musical relationships that can be recognized by listeners who understand the style being used. A curriculum for computer-assisted instruction in melodic composition should teach the proper use of musical patterns. However, drill and practice techniques alone cannot effectively teach both pattern construction and creative thinking. An effective computer teaching system should understand the structure of a student's memory well enough to know what patterns the student might create. The curriculum software also should be able to recognize patterns created by a student and evaluate whether or not these patterns are used in effective contexts.

Any compositional task is really a task in completing a pitch-time structure. Some elements are given or recalled. Others are generated or created. In the end, an event structure is assembled by the composer. A general definition of such an event structure follows:

E(vent group) = {E_1, E_2,...E_n}

E(vent)$_i$ = {P, Dur, Str, M, L, A}

P(itch) = <scale degree>|<note name>
Dur(ation) = <whole note>|<half note> etc.
Str(ess) = ∧ | / | ∪
M(ember of set) = TRUE FALSE
L(evel) = <event level>|<macro level>
A(ssociation) = HAPPY|POWERFUL|NULL
I(nterval) = {Dir, Size}
Dir(ection) = UP|DOWN
Size = {Gross, General, Specific}

Gross = BIG|SMALL
General = 2ND|3RD|4TH etc.
Specific = MAJ|MIN|PERF etc.

With these definitions, other musical elements can be defined as sets of incomplete musical events. For example, shape is

Shape = {Dir (I (E_1)), Dir (I(E_2)),...}

Figure 1 shows several student melodies that were composed as event structures to complete the motive marked "A."

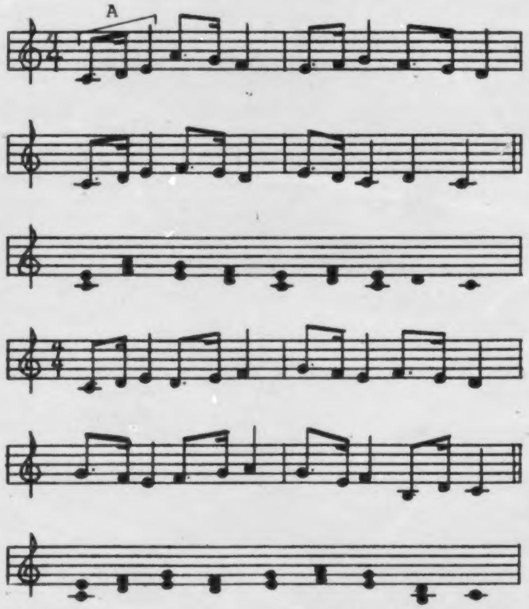


Figure 1. Structure of Student Melodies.

In these melodies, students held some features of the motive fixed but varied others. By using the third as an interval frame for pitch groups throughout the melody, the students exhibit a compositional strategy: they write sets that move by simple step relationships. The use of sets suggests that students work on more than one hierarchical level when they compose. They select a time and pitch frame, work within it, and then relate the frames by melodic intervals that usually are very small.

Observing students' strategies suggests the materials and procedures that a melodic composition curriculum should present. However, because machine recognition of patterns is difficult to implement when contexts vary, the melodic composition curriculum discussed in this paper does not recognize and evaluate students' patterns. Computers have no trouble finding what patterns might occur. However, as the author describes elsewhere, an algorithm for parsing patterns from the input of human subjects must use the differences that individual subjects perceive as delimiters (Dworak, 1977).

If a computer program is to recognize such individual differences, it must model a subject's memory. This paper later will present a general model for pitch relationships that define the major diatonic scale. The model has limitations, but in the course of its development, a number of memory models were studied. This research revealed that elaborate brain-memory models cannot be implemented effectively on existing computer architectures. The model and the strategies used in the melodic composition curriculum represent trade-offs between what could be implemented and what could effectively enhance student production.

A BRIEF SURVEY OF MEMORY MODELS

Models of memory have attempted to answer one or both of these questions:

1) What strategies enable a machine to produce the same structures that a human will produce?
2) What do human strategies suggest about the operations and limitations of the brain?

Early studies in decision-based human behavior recognized that a subject's behavior depended on the conditions that he faced. The general model

IF <condition> THEN <action>

gave rise to rule-based memory models. Rule-based models can be represented by lists or trees (Newell and Simon, 1972) or by digraphs and their matrices (Harray. Norman, and Cartwright, 1965). Laske applied rule-based memory models to the compositions of musical structures (Laske, 1977). However, composition is a poorly-specified task. Modeling it with rules is difficult. Such models work more successfully in well-specified game problems like chess (Frey, 1977).

Although condition-action models can represent what a person might do, they do not effectively represent what he will do. The digraph can be modified to represent an individual's preference for actions under specific conditions (Kandel and Lee, 1979). Individual preferences should not be confused with probabilities. Fuzzy set theory correctly shows that human behavior is not the product of random processes.

Rule-based memory models do not specify how or where the brain stores the rules that govern the mind's operation. Recent evidence suggests that memory is distributed throughout the brain (Thatcher and John, 1977). If this is true, memory fixation and retrieval do not depend upon the function of individual cells but upon the chemistry of all cells. RNA may be part of the chemical basis for the memory trace because it has semiconductor properties that could control the internal electrical functions of nerve cells (Brillouin, 1966; Hydén, 1970; Hofstader, 1979).

Distributed-memory models suggest that memory has features of a hologram (Pietsch, 1981). Holograms represent a scene by storing interference patterns or phase differences between a reference wave and waves scattered by objects in the scene. O'Keefe and Nadel demonstrate that the behavior of individual cells in a rat's brain is a function of both the environmental stimulus and the phase of an internal reference wave, the hippocampal theta rhythm (O'Keefe and Nadel, 1978).

A VECTOR-FIELD MODEL OF MEMORY SCANNING

An attractive feature of the hologram model is that it allows the storage of many memory traces with different phase angles in the same memory space. The proper recall then can be cued by the proper phase angle (Françon, 1974). This cuing is done by scanning the memory space. The destinations reached during a memory scan would determine a subject's behavior in much the same way that matrices of preferences would suggest what a subject's next action might be in a fuzzy set model.

The memory for pitch relationships that a composer might use can be modeled as a three-dimensional vector field in which memory elements occupy points in space (like a planetary system) and exert forces upon one another. The vector field is a useful model because it can represent both the pitch relationships that a composer might employ and the degree to which an individual has learned a particular relationship. Weak relationships might account for errors that a student might make if his contexts had not grown to match an ideal norm.

If a hologramic model of memory is valid, a three-dimensional vector field may be a valid scanning strategy. Traveling through the vector field may represent moving through phase angles that trigger different memories. The existence of such memory contexts again suggests that recall is not random but depends upon the state of an individual's memory at every point in time.

A CURRICULUM FOR MELODIC COMPOSITION

Implementation of a creative and thinking memory model on existing computers is not feasible because the computation that is required to scan such a model cannot be done in real time. In computer-assisted instruction, the computer's response to a subject's input must be given quickly. As a result, the melodic composition lessons in this curriculum are limited to:

1) The definition and presentation of commonly-used melodic patterns.
2) The description of operations that a student might employ to create melodic patterns.
3) Aural reinforcement of these patterns, both through ear training and through the performance of student's melodies.

The melodic composition lessons follow the limitations specified for first-semester freshman music theory students at North Texas State University:

1) Melodies consist only of the seven notes of the major diatonic scale.
2) Rhythmic values are no shorter than a simple division of the prevailing metric unit, i.e., in $\frac{4}{4}$, no shorter than an eighth note, in $\frac{3}{2}$, no shorter than a quarter note.
3) The seven scale members are represented in three sets (chords):

Tonic	{1, 3, 5}
Dominant	{5, 7, 2, 4}
Subdominant	{4, 6, 1}

Melodies from the first seven chapters of the Ottman Music for Sight Singing were studied to draw some conclusions about the kinds of set operations and relationships that should be taught in a curriculum (Ottman, 1967). Melodies in these chapters follow the constraints mentioned above.

Pitch patterns are composed by scanning a memory that contains representations of these sets or chords. This scanning represents the recall of pitch motions from memory. Diatonic tonal melodies allow three kinds of motion:

1) In-set: Within the immediate context, only members of a single set (chord) are used. For example, movement within the tonic would be 1-3-5 or 5-3-1.
2) In-set, interrupted: Immediate motion is controlled by one set, but this motion includes members of other sets. For example, motion from 1-3 could be embellished as 1-2-3 or as 1-4-3.
3) Inter-set: Immediate motion is controlled by one set, but the goal of this motion is a member of another set. For example, from dominant to tonic, the pitch pattern 7-2 resolves to 1, or the pattern 5-7-2-4 resolves to 3.

Most of these lesson functions can be presented by using drill and practice techniques. These techniques are supplemented by a graphics score editor and sound generating hardware, which allow the student to create and perform melodies. A pattern-generator subroutine allows the student to create and embellish sets and to hear melodic fragments generated by operations that the lessons suggest.

PATTERNS OF PITCH RELATIONSHIPS

In the curriculum, the pattern generator subroutine creates set patterns under the student's control. However, in order to better understand pattern generation, a vector-field model of musical thought was implemented as a separate routine, outside the curriculum. In this abstract model, memory relationships are defined as a function of three-dimensional space. As stated earlier, this space is similar to a three-dimensional planetary system or to the phase space that a hologramic model would need for recovering images. Travel through such a system depends on the forces exerted by the entire system.

When applied to a memory system, this model provides a method for scanning the memory and for recalling items from it. Of course, this abstract three-dimensional space must be filled with data, and this data must be built into a structure of relationships. Within the restrictions on pitch organization established earlier, musical relationships that can be described will suggest a structure for this model. Consider these pitch patterns:

1) Sets may be composed of only three or four members. Triangles may represent four-member sets. In-set motion in a three-member set might be modeled by scanning the triangle.
2) In-set motion can be interrupted by motion to another set. Motion from one set member to another with its interruptions can also be represented by triangles and tetrahedrons. Ideally, such interruptions leave the motion within the original set undisturbed (see Figure 2).

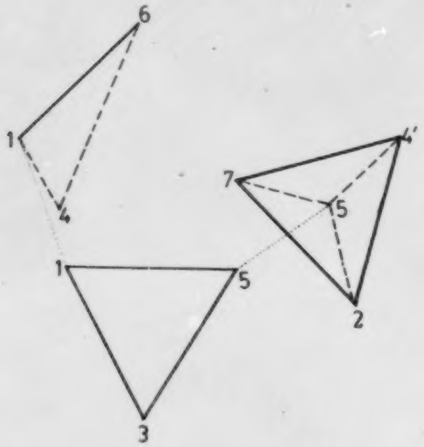


Figure 2. Triangular and Tetrahedral Relationships.

For the model under discussion, a geometric structure has been devised which allows all possible moves within the established limits of melodic motion. The structure is shown in Figure 3. This figure is an icosahedron. Each half of the figure represents the relationships between scale degrees 1 and 5 and the remaining five scale degrees. The second half of the figure, a rotated mirror image of the first half, represents the relationships among the remaining five scale degrees.

This geometric arrangement makes any scale degree the closest neighbor to all other scale degrees so that any interval can be represented by a direct path between two points in the model. The geometric arrangement is not significant in itself. However, it suggests that tetrahedral relationships within sets and with their embellishments can form a larger, closed system of relationships. In addition to representing the types of relationships that can occur between scale degrees, the lines connecting nodes should be imagined to be relationships of varying strengths, some learned poorly, others learned well.

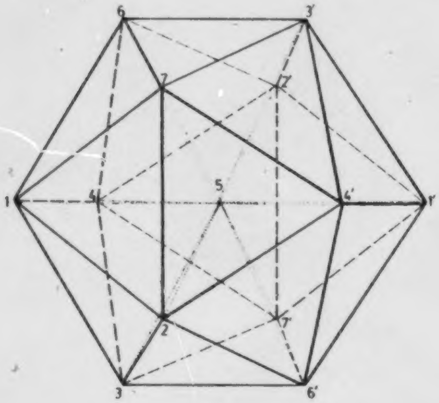


Figure 3. Icosahedral Model of Diatonic Pitch Memory.

BIOLOGICAL FOUNDATIONS OF MEMORY

The symmetry of this model resembles crystalline structure. Crystal structure theory provides insight into the forces that operate in such a geometric system and into how such a system might be scanned. Thatcher and John suggest that context memory is an ionic memory that dynamically changes its force structure in response to the nerve cell's electrical and chemical activity.

Biochemical research suggests that water crystals may be instrumental in providing the energy changes and the structures needed by a context-based memory system (Szent-Györgyi, 1957). Stillinger describes the crystal structures of water and points out that water assumes certain loose geometric structures, even in the liquid state (Stillinger, 1980). He also demonstrates that water crystals resembling those of ice may form in the presence of organic molecules, even at body temperature.

Water crystals are also believed to be part of the nucleic acid structures in the brain cells. The geometries of water crystals surrounding RNA are not known, but many orientations are believed to be possible (Scordamaglia et al, 1977; Clementi et al, 1977). RNA molecules are very large, providing many possible orientations. Perhaps water structures, because of their ionic and crystalline qualities, provide the electrical field changes that are necessary for recall. The system may, in fact, be bidirectional: specific crystal orientations may be created in response to electrical discharges within the cell. The basic proposition that emerges from these assumptions is that water crystal structures are responsible for the ionic fields needed for context-based memory structures, and that such structures, like the memory that they represent, are transient in nature. If this were true, some aspects of Thatcher's ionic memory would be satisfied. Because the crystal structures could be built in response to external electrical events and in relation to indestructible internal chemical structures, recall of the same experience could be repeated indefinitely. This argument is supported by evidence that vasopressin, a chemical produced by the brain, both causes the body to retain water and enhances recall (Weingartner et al, 1981).

An important feature of water crystals is that, in all their configurations, they form tetrahedral bonds, i.e., each molecule is surrounded by four nearest neighbors. New research suggests that short-term memory is limited to the storage of four or five items, not the seven suggested in earlier literature (Starkey and Cooper, 1980; Ericsson, Chase and Faloon, 1980).

The limitation of consciousness and context to four or five items can be verified easily. Simply improvise a melody, singing it. Be conscious of your own awareness, and judge the limit of your awareness. The immediate context (near past, near future) will be about four or five pitch events, i.e., three or four intervals.

SUMMARY OF PROGRESS AND CONCLUSIONS

Five melodic composition lessons currently are implemented on a Motorola 6809 microprocessor and are being used by a group of about twenty-five freshman first-semester music theory students at North Texas State University. These lessons provide the students with practice in recognizing sets of scale degrees. They explain both the motions that are possible within sets and how these motions may be embellished. Students are asked to identify set motions, and they can use these motions to create melodic patterns to which they may listen.

The lessons stress the variety of intervals that result from simple set motions. Ear-training in interval recognition is a major part of three of the lessons because well-learned interval relationships are needed for the successful composition of melodic structures. Throughout the lessons, aural reinforcement of correctly-named intervals and of correctly-formed sets encourages intervalic variety in student melodies.

Finally, the lessons enable the students to generate and listen to melodic patterns, to write melodies on a staff, and to hear and edit the melodies that they compose. Because machine evaluation of melodies is impossible, the curriculum driver stores all editorial changes that a student makes in a melody for later human evaluation.

Two pilot groups have used this curriculum since November, 1980. A final subject group currently is participating in this research. A melodic composition pretest and the Barron-Welsh Revised Art Scale were administered both to this subject group and to a control group during the last week of March and the first week of April, 1981. The Revised Art Scale is a standard measure of creative potential. Post-tests for both groups were administered during the first week of May, 1981. An evaluation of the success of the curriculum will be available in June. 1981.

In summary, melodic composition CAI cannot successfully be done using only drill and practice techniques. The memory modeling that is required reveals many facets of musical memory. Models of memory and of creative thought should continue to be developed in future curriculua in music CAI. Models of brain functions also suggest new computer architectures. Architectures that simulate learning and creative thought will permit exciting new interactions between humans and machines in a creative learning environment.

ACKNOWLEDGEMENTS

This research was funded by National Institute of Education grant G-80-0012. Facilities were provided by the North Texas State University School of Music and the Faculty Research Committee. Special thanks must go to Renee McCachren, who designed most of the icosahedral pitch model and most of the lessons. She was assisted in lesson design by Eileen Cloutier. Philip Baczewski designed and implemented the graphics editor and the ear-training routines and he contributed to the implementation of other software.

Steven Bertsche designed the driver that is used for this curriculum, and he suggested formats for the implementation of many of the lessons' functions.

REFERENCES

Brillouin, Leon. "Giant Molecules and Semiconductors," in de Broglie, Louis, ed. Wave Mechanics and Molecular Biology. Reading, MA: Addison Wesley Publishing Co., 1966.

Clementi, E., Cavallone, F., and Scordamaglia, R. "Analytical Potentials from 'ab Initio' Computations for the Interactions Between Biomolecules. 1.Water with Amino Acids," Journal of the American Chemical Society, Vol. 99, No. 17 (1977), pp. 5531-5545.

Dworak, Paul E. The Utility and Compositional Ramifications of Grouping Theory. Ann Arbor, MI: University Microfilms. 1977.

Ericsson, K. A., Chase, W. G., and Faloon, S. "Acquisition of a Memory Skill," Science, Vol. 208, No. 4448 (1980), pp. 1181-1182.

Françon, M. Holography. New York, NY: Academic Press, 1974.

Frey, Peter W., ed. Chess Skill in Man and Machine. New York, NY: Springer-Verlag, 1977.

Harray, Frank; Norman, Robert A.; and Cartwright, Darwin. Structural Models: An Introduction to the Theory of Directed Graphs. New York, NY: John Wiley and Sons, 1965.

Hofstader, Douglas R. Gödel, Escher, Bach: An Eternal Golden Braid. New York, NY: Basic Books, 1979.

Hydén, Holger. "The Question of a Molecular Basis for the Memory Trace," in Pribram, K. and Broadbent, D., eds. Biology of Memory. New York, NY: Academic Press, 1970.

Kandel, Abraham and Lee. Samuel C. Fuzzy Switching and Automata: Theory and Applications. New York, NY: Crane, Russak, and Company, 1979.

Laske, Otto. Music, Memory, and Thought. Ann Arbor, MI: University Microfilms, 1977.

Newell, Allen and Simon, Herbert A. Human Problem Solving. Englewood Cliffs, NJ: Prentice-Hall, Inc. 1972.

O'Keefe, John and Nadel, Lynn. The Hippocampus as a Cognitive Map. Oxford: Clarendon Press, 1978.

Ottman, Robert. Music for Sight Singing. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1967.

Pietsch, Paul. Shufflebrain. Boston, MA: Houghton, Mifflin Co., 1981.

Scordamaglia, R., Cavallone, F., and Clementi, E. "Analytical Potentials from 'ab Initio' Computations for the Interactions Between Biomolecules. 2. Water with the Four Bases of DNA," Journal of the American Chemical Society, Vol. 99, No. 17 (1977), pp. 5545-5550.

Starkey, P. and Cooper, R. J., "Perception of Numbers by Human Infants," Science, Vol. 210, No. 4473 (1980), pp. 1033-1035.

Stillinger, F. H. "Water Revisited," Science, Vol. 209, No. 4455 (1980), pp. 451-457.

Szent-Györgyi, A. Bioenergetics. New York, NY: Academic Press, 1957.

Thatcher, Robert W. and John, E. Roy. Foundations of Cognitive Processes. Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers, 1977.

Weingartner, H. et al. "Effects of Vasopressin on Human Memory Functions," Science, Vol. 211, No. 4482 (1981), pp. 601-602.

Computer-Assisted Instruction in Music:
Analysis of Student Performance 1973-1980

Wolfgang Kuhn
Stanford University

and

Paul Lorton, Jr.
University of San Francisco

INTRODUCTION

Drill and practice computer-assisted instruction (CAI) programs have been used in various curriculum areas to study the acquisition of skills. For the past several years, a CAI program in music has been used at Stanford University. This paper describes and surveys the data analysis performed on the item-by-item response data generated by the CAI music program.

The CAI-music program has been described extensively elsewhere (see Lorton, Killam & Kuhn, 1975 and Lorton, Killam & Kuhn, 1981). Briefly, a solid state Thomas organ is connected to the PDP-10 timesharing system operated by the Institute for Mathematical Studies in the Social Sciences (IMSSS) at Stanford University. The text portion of the curriculum is presented and the student enters answers on a cathode ray tube (CRT) display next to this organ. Under program control, the organ plays the musical examples presented in the drill and practice curriculum. This system has been used as an adjunct to undergraduate music courses taught at Stanford since spring quarter, 1973.

There is no question that music exists primarily as an enjoyable, aesthetic experience. Nevertheless, in the preparation for a career in music as a teacher, composer, or performer, some basic disciplines are essential to pre-professional career training: training ear and eye coordination, performance ability, and detailed drill and practice exercises. Computer-assisted instruction in music facilitates training in basic skills, specifically focussing on ear and eye coordination.

In addition to its value as an instructional tool, one of the main purposes of this system is to collect data on student performance in the various strands of the curriculum. Extensive and detailed data have been collected and analyzed. One reason for the data collection is to study students to better understand the individual differences associated with learning basic music theory skills. Another reason is to help revise and extend the curriculum. As an ongoing process, the analysis of student response and use information from the CAI provides the kind of information useful for monitoring the effect of the curriculum and for planning future extensions, such as the microcomputer project which will be discussed later.

STUDENT POPULATION USING CAI

Participating students have been from a variety of undergraduate music classes offered by the Stanford Music Department. The greatest number of students have come from the first-year music theory courses, a three-quarter sequence. This sequence is required of all undergraduate music majors (open to non-music majors as class space permits) and can begin in any quarter, so that drill and practice in elementary ear training is used throughout the year. Some students must delay continuing the sequence because of schedule conflicts; so, some resume their theory coursework after a period of minimal use of their acquired skills in music theory. The CAI program refreshes their ear training skills.

Although CAI scheduling preference is given to class members in the first-year courses, students in the second-year curriculum (another three-course sequence) also use the CAI program, as time permits. In addition, the music department administers required examinations in ear training, independent of course work, to qualify music majors for upper division work. Considerable use of the CAI program has been made by those students who have not yet passed their departmental examinations.

In summary, although the greater portion of students using the CAI program are music majors in their first year's courses, the program has proved attractive and valuable to undergraduates at differing course levels.

To date, about 440 students have used the system for an average of 7.2 sessions each, generating about 3100 bytes (characters) of data per session. In all, over 10 million bytes of data have been recorded. Table 1 summarizes the quarter by quarter use of the system. The fall quarter usually shows the heaviest use.

Table 1 Histogram of Usage by Academic Quarter

Quarter	Sessions		
	10 100 200 300	ssn/day	min/day
SPRING 1973	*********************	4.38	104.23
FALL 1973	**************************	5.56	95.35
WINTER 1974	******************	4.40	78.20
SPRING 1974	*********************	4.61	99.95
FALL 1974	*************************	6.68	155.65
WINTER 1975	*************	3.64	92.26
SPRING 1975	***********	2.88	77.97
FALL 1975	*********************	4.80	120.77
WINTER 1976	**********	2.78	85.90
SPRING 1976	**	1.38	29.82
FALL 1976	********************	6.70	137.87
WINTER 1977	************	4.32	97.19
SPRING 1977	***	1.94	30.11
FALL 1977	************************	5.72	131.74
WINTER 1978	*********************	6.38	165.13
SPRING 1978	*******	3.08	107.11
FALL 1978	************	2.93	70.24
WINTER 1979	*******	2.09	56.82
SPRING 1979	**	1.40	41.66
FALL 1979	*************	3.53	104.55
WINTER 1980	***	1.65	49.79
SPRING 1980	**	1.44	15.71
FALL 1980	*********	2.63	69.50

CURRICULUM OVERVIEW

The curriculum presented by this system is divided into several strands by content and by the nature of the training tasks. Strands include intervals, triads, melody, rhythm, chords, and chord identification. To clarify the ensuing discussion, it may be well to keep the following definitions of musical terms in mind:

Intervals - the relationship of two pitches or tones. In this program, we are limited to diatonic intervals, i.e., those using two different letter names. The possible intervals are:

Name	written	example
Minor Second	MI2	C,D flat
Major Second	MA2	C,D
Minor Third	MI3	C,E flat
Major Third	MA3	C,E
Perfect Fourth	PE4	C,F
Augmented Fourth	AU4	C,F sharp
Tritone	T	C,F sharp
Diminished Fifth	DI5	C,G flat
Perfect Fifth	PE5	C,G
Minor Sixth	MI6	C,A flat
Major Sixth	MA6	C,A
Minor Seventh	MI7	C,B flat
Major Seventh	MA7	C,B
Perfect Octave	PE8	C,C

Melody - in the broadest sense, a succession of musical tones, as opposed to harmony.

Rhythm - measurement of time, resulting from various note lengths, configurations using patterns of notes or rhythmic variations. Rhythm is also related to accents, meter and musical pulse or beat.

Triads and Chords - refer to the simultaneous sounding of tones. Specifically, triads consist of three notes and chords of more than three. Triads are classified as minor (MI), major (MA), diminished (DI) or augmented (AU). Chords are identified by roman numerals, referring to the scale degrees which are used as roots or lowest notes. Primary chords are I, IV, V; secondary chords are II, III, VI, VII.

INTERVAL STRAND

The interval strand material is divided into two segments. The first gives students practice in interval identification; the second gives students practice in spelling the pitches which make up the intervals. Both segments contain intervals no larger than the octave. The program generates the intervals within each topic and presents them in random order. Also, it randomizes the mode of presentation as ascending, descending, or simultaneous. The program checks student performance after each set of ten intervals. If the student has achieved more than 80% accuracy within the ten intervals, the student is informed of his total percent correct and offered the option to proceed to the next topic. (Students may change topics or strands at any time, but must request the change.)

Information on all of the foregoing is collected and stored, e.g., the interval presented, the pitches of which the interval is composed, and its mode of presentation.

TRIAD STRAND

The triad strand consists of two segments. The first gives students practice in identifying major, minor, augmented, and diminished triads in root position. The second segment provides practice in identifying inversions of these triads.

MELODY STRAND

The melody strand also contains two segments. The first (topics 1-5) is designed for pre-college students. To conform with Kodaly System of Music Education answers are given in abbreviations of moveable "do." This segment of the melody strand has been little used by college students. The second segment (topics 6-11) contains short melodies of graduated difficulty. Only topics 6-10 are of general use to students for drill and practice in the first two years of music theory; topic 11 contains a more experimental and complex series of melodies. Topics 6 through 10 consist of five melodic fragments each. The student is given the key and the tonic chord. He enters his answer in pitch names; sharps and flats are indicated by "+" or "-."

The curriculum is not generated by the program, although the program randomly transposes the examples so they may be presented in a key other than that indicated. When the example is transposed, the opening tonic chord is also transposed, so that the transposed examples are of no greater difficulty for students unless they have pitch recognition. The transposition feature was initiated to make those few students with pitch recognition deal with melodic patterns in intervallic terms, rather than allowing them to answer the examples on a "pitch-only" basis.

Since the examples are not generated, they remain the same. Repeating the examples has minimal instructional value. The data accumulated allow exact pitch of student errors to be calculated, as well as the point of error within the melodic example.

RHYTHM STRAND

The rhythm strand is composed of eight topics presenting rhythmic patterns of graduated difficulty. Each rhythmic pattern is presented by notes in a simple melodic pattern, and is preceded by a series of notes to give the basic rhythmic pulse for the example.

The rhythm strand has proven to be of limited value in its present form primarily due to hardware limitations which do not control amplitude of notes. Consequently, rhythmic examples without essential accents sound only the duration values of the notes. Students answer in letter abbreviations of the rhythmic values of the notes: W=whole note, H=half note, Q=quarter note, T=eighth note, and X=sixteenth note. The last two abbreviations were chosen to be other than the first letter of the name of the note value because "E" is used as a pitch symbol and "S" is used as an abbreviation for "sol."

The examples have proven difficult because of the mode of presentation and, consequently, little student use has been made of this strand and less data are available from it.

CHORD STRAND

The chord strand is constructed of seven topics dealing with chord progressions of graduated difficulty. The first three topics deal with identification of the tonic and dominant chords and authentic cadences. Subsequently, the remaining diatonic chords are introduced; topic 6 introduces inverted chords, and topic 7 is composed of chord progressions in minor keys. Answers are roman numerals for the chords. Attention is focused at first on the primary triads of I, IV, and V. Chords contained in the progression other than those to be identified are indicated by the student as a dash. Precise student identification of chords other than I, IV, and V is introduced in topics 4 and 5. Students may repeat the examples as often as necessary (this is true on all of the strands) before entering their answers. Students made more frequent use of the repetition option on the chord strand.

The chord strand has been a popular one with students, and considerable data have been collected from it. These data can be analyzed for specific content errors and for position of errors within the chord progression.

CHORD IDENTIFICATION STRAND

The chord identification strand is designed for more advanced students. It is composed of three segments: the first asks for the quality, inversion, and Roman Numeral of a triad in relation to a given tonic; the second requires the same identification process, but the chord presented is a seventh chord; the third is a series of chords in various inversions (both triads and seventh chords) to be spelled by the student with only the bass note given.

The first two segments require accuracy of more than 75% before the student can progress to the next topic. Although the program does not generate the curriculum, it accesses the examples randomly from a wide range of material. Therefore, students can repeat topics without receiving material in the same order. Also, the musical key in which the example is played is randomized. The final segment, that of chord spelling, does not have curriculum generated by the program.

In practice, this topic has proved so difficult that few students have reached the third segment. The first two segments have received fairly heavy use, and considerable data are available for analysis. Data analysis can show student error in relation to each of the areas of the required answer: chord quality, chord inversion, or roman numeral, giving insight as to where in the chord recognition process students are most likely to err.

Table 2. Histogram of number of sessions by number of Students

FALL 1978	number of students	
# of ssns	1 . 5 . 10 15 20	minutes/ssn
1	********************	21.50
2	******	32.44
3	********	34.73
4	**	16.43
5	**	36.91
6		.00
7	**	23.80
8	**	21.89
9		.00
10	******	26.99
11		.00
12	**	28.39
13	**	34.26
14		.00
15+	**	15.21

TOTALS: Students 27 Sessions 127
SSN/Student 4.70 Minutes/SSN 26.13

FALL 1979	number of students	
# of ssns	1 . 5 . 10 15 . 20	minutes/ssn
1	************	28.98
2	******	34.99
3	**********	31.36
4	********	24.27
5		.00
6	******	34.61
7	**	29.38
8,9		.00
10	******	35.98
11		.00
12	**	46.72
13,14		.00
15+	**	33.55

TOTALS: Students 27 Sessions 130
SSN/Student 4.81 Minutes/SSN 33.71

FALL 1980	number of students	
# of ssns	1 . 5 . 10 . 15 20	minutes/ssn
1	**********	19.59
2	**	23.20
3	****	35.67
4	****	49.55
5	**	24.04
6	**	26.48
7	****	42.98
8		.00
9	**	19.41
10,11		.00
12	**	32.04
13,14		.00
15	**	23.54

TOTALS: Students 17 Sessions 82
SSN/Student 4.82 Minutes/SSN 31.07

PRODUCTS OF THE DATA ANALYSIS

Data are continually collected as the students interact with the program. The results reported here are for data collected between March 1977, and June 1980. Because use is heaviest in the fall quarter, particular emphasis will be given to the fall quarters 1978, 1979, and 1980.

Table 2 shows the number of students taking various sessions for these three quarters. The greatest single number of students took just one session. About three quarters of the students did use the system more than once. For all three fall quarters, the average number of sessions (mean) was about five, and the average length of each session was about 30 minutes.

INTERVAL STRAND

The interval strand has been the most heavily used, especially in the fall quarter. This activity, recognizing two note intervals, is fundamental to ear training and music theory and is an activity on which most students, new to ear training, need much drill and practice.

As a special application of the interval strand, an experiment was conducted replicating an earlier non-CAI study. The experiment explored some of the parameters in interval identification by students who had successfully completed the first segment of the interval strand. The experiment presented simple intervals in ascending, descending, and simultaneous forms at both .1 and .2 sec., thus totalling six topics. Each topic contained an equal number of intervals constructed of the same pitches, but order of presentation within topics was randomized. The results of this study are discussed at length elsewhere (Killam, Lorton, and Schubert, 1975).

Table 3 shows the performance on the interval strand for the last three fall quarters (1978, 1979, 1980). The last two rows of numbers for each quarter are the most informative. The "% C" row is the percent correct. The "% R" row is an index of the number of times the student requested a repeat of the stimulus. The two notes are presented and an identification of the interval is requested. If a student wishes to hear the notes again, another presentation of the stimulus can be requested.

For all three quarters, although the actual percentages vary widely, intervals played in ascending order were correct most often and were repeated least often. The most difficult intervals where those presented simultaneously. The percent correct and the number of repeats requested both reflect this. For interval quality, perfect intervals were identified correctly more often and repeated least often. Major and minor intervals were about equally difficult. The other interval qualities occurred much less frequently than these three main types and showed relatively large fluctuations in both percent correct and proportion of repeats.

TRIAD STRAND

The triad strand is the second most used. Identification of three note combinations is a usual step after the two note intervals have been addressed. This strand has also been the topic of earlier systematic studies (see Killam, 1976). In addition, the studies of intervals and triads have been compared elsewhere (see Lorton & Killam, 1976).

In Table 4, the performance on the triad strand for the last three fall quarters is presented. Again, for each of the three quarters, the pattern remains the same while the specific numbers vary. Ascending triads are more correctly identified and repeated less often. Triads played simultaneously are repeated more often and identified less accurately. In general, major triads are more correctly identified than triads

Table 3: Performance on INTERVAL Strand
FALL 1978, FALL 1979, FALL 1980

FALL 1978										
	Total	Ascend	Desc.	Simul	MA	DI	AU	PE	MI	T
Cor	1860	541	887	353	535	3	5	310	482	82
Err	1031	206	437	382	323	5	2	128	412	33
Rep	971	143	307	356	283	4	1	106	332	38
% C	64.3	72.4	67.0	48.0	62.4	37.5	71.4	70.8	53.9	71.3
% R	33.6	19.1	23.2	48.4	33.0	50.0	14.3	24.2	37.1	33.0
FALL 1979										
	Total	Ascend	Desc.	Simul	MA	DI	AU	PE	MI	T
Cor	1274	336	626	248	391	4	0	284	348	67
Err	915	187	460	260	299	3	1	165	330	57
Rep	995	144	439	275	262	5	1	174	325	65
% C	58.2	64.2	57.6	48.8	56.7	57.1	.0	63.3	51.3	54.0
% R	45.5	27.5	40.4	54.1	38.0	71.4	100.0	38.8	47.9	52.4
FALL 1980										
	Total	Ascend	Desc.	Simul	MA	DI	AU	PE	MI	T
Cor	1153	355	576	221	359	2	6	224	398	37
Err	433	89	206	138	159	3	6	68	174	6
Rep	1521	323	626	513	519	5	19	244	537	79
% C	72.7	80.0	73.7	61.6	69.3	40.0	50.0	76.7	69.6	86.0
% R	95.9	72.7	80.1	142.9	100.2	100.0	158.3	83.6	93.9	183.7

which represent other qualities. Although, because of some restructuring of the curriculum in the most recent quarter, diminished triads approach major triads in accuracy of identification, the former are still repeated more often than the latter.

Table 4. Performance on TRIAD Strand
FALL 1978, FALL 1979, FALL 1980

FALL 1978	Total	Ascend	Simul	MA	DI	AU	MI
Cor	184	99	85	77	33	3	71
Err	73	28	44	30	9	3	30
Rep	49	13	36	15	8	3	23
% C	71.6	78.0	65.9	72.0	78.6	50.0	70.3
% R	19.1	10.2	27.9	14.0	19.0	50.0	22.8

FALL 1979	Total	Ascend	Simul	MA	DI	AU	MI
Cor	411	240	164	162	112	11	125
Err	186	57	110	58	50	4	73
Rep	191	36	123	52	43	2	75
% C	68.8	80.8	59.9	73.6	69.1	73.3	63.1
% R	32.0	12.1	44.9	23.6	26.5	13.3	37.9

FALL 1980	Total	Ascend	Simul	MA	DI	AU	MI
Cor	149	73	70	50	52	9	38
Err	23	9	12	10	9	0	4
Rep	86	17	47	21	46	1	7
% C	86.6	89.0	85.4	83.3	85.2	100.0	90.5
% R	50.0	20.7	57.3	35.0	75.4	11.1	16.7

Since the interval and triad strands have had the most use over the years, they are providing the best information for improving the curriculum. Certain of the exercises in these strands have been restructured to present items of greater difficulty in conjunction with those items with which they are most often confused. Much of this restructuring has taken place in the last few months and will be evaluated as part of the current effort to implement this curriculum on a microcomputer-based system.

STRAND-BY-STRAND PERFORMANCE

Table 5, 6, and 7 summarize all the problems presented in the last three fall quarters on a strand-by-strand basis. The order of the tables represents a restructuring of the basic curriculum begun in the fall quarter 1980. For this reason some strands, such as scales and kodaly, which were only recently added have had no instructional use by the students. For others, such as the rhythm strand, changes in the basic sequence of the curriculum and findings of earlier analysis have compelled suspending their presentation until new hardware is added to the music CAI system as a result of the current effort in transferring the program to a microcomputer system.

The advanced strands, such as chords and chord identification, show less use than the three most frequently accessed strands. This is, in part, due to earlier findings which have guided the specification of new hardware that will allow a better implementation of these strands through a more flexible sound source and through input of responses via a musical keyboard. The results of the analyses which assisted in this effort will be reviewed for the rhythm and chord identification strands.

TABLE 5. Strand by Strand Problem Counts - Fall 1978.

Strand	Total	Cor	Err	Rep	% C	% R
PRELIMINARY	283	262	21	96	92.6	33.9
SCALES						
INTERVALS	5559	3427	2132	2138	61.6	38.5
TRIAD	454	322	132	94	70.9	20.7
KODALY						
MELODY	182	139	43	27	76.4	14.8
RHYTHM	3	2	1	1	66.7	33.3
COUNTER-POINT						
HARMONY						
MODULATION						
CHORDS	10	8	2	0	80.0	.0
CHORD-I.D.	52	40	12	6	76.9	11.5
STRUCT (old)	4	2	2	1	50.0	25.0
TOTAL	6547	4202	2345	2363	64.2	36.1

TOTAL DAYS 43 MINUTES PER DAY 71.736

TABLE 6. Strand by Strand Problem Counts - Fall 1979.

Strand	Total	Cor	Err	Rep	% C	% R
PRELIMINARY	233	207	26	70	88.8	30.0
SCALES						
INTERVAL	5346	2674	2672	3145	50.0	58.8
TRIAD	1159	745	414	372	64.3	32.1
KODALY						
MELODY	195	140	55	71	71.8	36.4
RHYTHM						
COUNTER-POINT						
HARMONY						
MODULATION						
CHORDS	196	116	80	67	59.2	34.2
CHORD-I.D.	78	40	38	25	51.3	32.1
STRUCT (old)	44	32	12	20	72.7	45.5
Total	7251	3954	3297	3770	54.5	52.0

TOTALS DAYS 38 MINUTES PER DAY 104.449

TABLE 7. Strand by Strand Problem Counts - Fall 1980.

Strand	Total	Cor	Err	Rep	% C	% R
PRELIMINARY	166	151	15	37	91.0	22.3
SCALES						
INTERVAL	3262	2264	998	3683	69.4	112.9
TRIAD	323	276	47	208	85.4	64.4
KODALY						
MELODY	6	5	1	3	83.3	50.0
RHYTHM						
COUNTER-POINT						
HARMONY	13	11	2	2	84.6	15.4
MODULATION	11	8	3	2	72.7	18.2
CHORDS	89	76	13	27	85.4	30.3
CHORD-I.D.	44	41	3	0	93.2	.0
Total	3914	2832	1082	3962	72.4	101.2

TOTAL DAYS 35 MINUTES PER DAY 69.410

RHYTHM STRAND

Although the rhythm strand is one of the content areas used less than others by students, some general conclusions can be made from the summary presented in Table 8.

Topic 1 is an introductory unit; requests for repetition of material were relatively higher as students accustomed themselves to the problem format. Topics 2 and 3 had the highest levels of accuracy. The topics were constructed of quarter and eighth notes only, and the problems contain from four to eight notes. Topics 4 - 7 introduce notes of longer values into the rhythm patterns, although the number of notes in the problems do not exceed eight notes. The percent correct in the problem counts above indicates that student accuracy, in general, decreased with the addition of other, longer note values, even though the number of notes in the problem grew no larger.

The rhythm strand data imply that the number of notes in the examples is not as important a factor in accuracy (within the limited parameters employed here) as is the variety of note values. This is all the more interesting when one considers that the shorter note values are presented in the first topics where student accuracy was higher. Student accuracy dropped as the longer note values of the half note and dotted half note were introduced.

Because the hardware could not control accents on the notes, both the rhythm and melody strands are awaiting better implementation on the new microcomputer-based equipment which will replace the current hardware.

Table 8. Problem by Problem Counts for the RHYTHM Strand.

Prb #	prbs	corr	reps	% corr	% reps
1.1	32	26	5	81.25	15.63
1.2	29	24	3	82.76	10.34
2.1	36	29	8	80.56	22.22
2.2	39	36	2	92.31	5.13
3.1	27	24	0	88.89	.00
3.2	21	21	0	100.00	.00
3.3	16	16	1	100.00	6.25
3.4	21	21	0	100.00	.00
4.1	23	16	4	69.57	17.39
4.2	22	0	0	.00	.00
5.1	30	25	10	83.33	33.33
5.2	20	8	6	40.00	30.00
5.3	14	12	0	85.71	.00
7.1	18	13	9	72.22	50.00

CHORD IDENTIFICATION STRAND

Table 9 summarizes student data on the chord identification strand, topics 1-10.

Topic 1 presents material introducing the answer format required by this strand. Topic 2 presents only the primary triads for identification. Topic 3 uses the II, III, and VI triads. The percentage of accuracy was nearly the same on both topics, within .25%; the percentage of requests for repetition was nearly the same. Accuracy exceeded by ten percentage points the 75% required by the program for successful completion of each topic.

Topic 4, presenting III, VI, and VII triads, had a 77% level of student response accuracy, or approximately the level required for successful completion. The total number of problems increased over that of Topic 3, indicating that students had to answer a larger number of problems to achieve the required level of accuracy. Requests for repetition increased by 33 percentage points.

Fewer students used Topic 5, which presents tonic, subdominant, and dominant seventh chords, but level of accuracy was quite high at 94% (more than 80% required for successful completion) and requests for repetition dropped from the level of topic 4.

Topics 6-10 present other seventh chords and received only about one quarter of the use of topic 5. Student accuracy ranged from 83-88%, or just above that required for successful completion.

The chord identification strand is more difficult than melody, rhythm, or chord strands, and is not recommended to students until they have completed substantial amounts in other content areas. Nonetheless, several students completed some portion of this strand. These students made heavy use of the segment devoted to identification of triads in relation to a given tonic. This showed some evidence of the difficulty identifying triads other than the primary ones, which was found in the chord strand.

Table 9. Problem by Problem counts for the CHORD IDENTIFICATION Strand.

Prb #	prbs	corr	reps	% corr	% reps
1.1	18	18	0	100.00	.00
1.2	20	19	0	95.00	.00
1.3	21	6	0	28.57	.00
2.1	638	544	264	85.27	41.38
3.1	385	330	154	85.71	40.00
4.1	431	332	318	77.03	73.78
5.1	198	187	125	94.44	63.13
6.1	59	52	23	88.14	38.98
7.1	49	41	7	83.67	14.29
9.1	70	60	48	85.71	68.57
10.1	30	25	20	83.33	66.67

PROSPECTS

This paper surveyed both the extensive nature of the data collected and analyzed in the CAI-music project and the uses of that information. Such analysis gives an additional dimension to CAI because it produces information that can be used both to assess student performance and to evaluate the success of the instruction. In this latter way data analysis can contribute directly to improvement of instruction. We have been able to identify some particularly difficult training problems and can design drill and practice exercises to help students master them. We also have a better understanding of how students acquire various skills under computer direction. The activities for various exercises have a data-based rationale.

The success of the CAI-music program has depended heavily on detailed response data collected. Even without these data, the CAI music program is a worthwhile instructional tool; but with these data, the program becomes a valuable research tool with potential for a significant impact on education.

The current focus of this project is, with equipment provided by the Apple Education Foundation, to adapt the Stanford system to a microcomputer. Many of the features sought in the development of this system grew out of the findings of the data from the original equipment. The rhythm strand needs elaborate control over the accents; hardware was acquired which allows this. Melodic examples need to be generated by the computer to provided a greater richness in the melody strand; software to meet this need is being developed.

CAI implemented on a microcomputer will include data retention and analysis routines; these will provide the evaluation tools necessary to perform studies of students and curriculum for those who make use of the Apple based system. It will also make the system affordable for either home or pre college school use.

REFERENCES

Killam, R. N. "A Study of Triad Recognition and Confusion by Stanford Undergraduate Music Students." Unpublished DMA Final Project, Stanford University, 1976

Killam, R. N., P. Lorton, Jr. & E. D. Schubert, "Interval recognition: Identification of harmonic and melodic intervals.", Journal of Musical Theory, 1975, 19, 212-234

Lorton, P. Jr., R. N. Killam & W. Kuhn. "Research on Computer-Assisted Instruction in Music." in P. Suppes (Ed.), UNIVERSITY-LEVEL COMPUTER ASSISTED INSTRUCTION AT STANFORD: 1968-1980. Stanford, CA: Stanford University, Institute for Mathematical Studies in the Social Sciences, 1981 (In Press)

Lorton, P. Jr., R. N. Killam, & W. Kuhn. "A Computerized System for Research and Instruction in Music.", In O. Lecarme & R. Lewis Computers in Education: Proceedings-IFIP 2nd World Conference. Amsterdam: North Holland Press, 1975. Pp 765-769

Lorton, P. Jr. & R. N. Killam. "Interval and Triad Recognition: A Comparison of Two Studies of Musical Perception." Paper presented to the Second Annual Meeting of the National Consortium for Computer-Based Musical Instruction, Minneapolis, MN, August 1976.

COMPUTERS AND CONTINUING HEALTH EDUCATION

Sponsored by The Society for Computer Medicine

ABSTRACT:

The continuing education of practicing health professionals has become, as it has for educators, a matter of growing national interest and concern, and the subject of increasing public scrutiny. While both the health and education professions have been making a considerable effort to change and improve (e.g., the number of participants in continuing health education has increased four-fold in the last ten years), the current system is simply not adequate to handle the projected need, especially as continuing education becomes mandated by legislation. Also, the current feeling about spending public funds makes it unlikely that we will see a mass infusion of public funds to expand the current system.

An increasing number of people have been turning their attention to the use of computers and telecommunications as the best hope for providing the capacity needed to meet the growing demand for continuing health education. These two technologies, used within appropriate professional structures, can deliver to the home quality education tailored to the needs of the individual health practitioner, while maintaining the records and generating the reports needed for credential purposes.

First, a brief overview of some of the key factors at work within continuing health education will be presented, followed by a paper discussing one of the major potential uses of the computer in continuing health education, the simulation of patient-management situations. The session will conclude with a panel and audience discussion of problems and potentials for the use of computers in continuing education as a whole.

Participants:

Richard E. Pogue
Health Systems and Information Sciences
Medical College of Georgia
Augusta, GA 30901

Ronald C. Comer
Division of Computer Services for Medical Education and Research
Ohio State University
Columbus, OH 43210

James E. Eisele
College of Education
University of Georgia
Athens, GA 30602

Lynn L. Peterson
Department of Medical Computer Science
The University of Texas Health Science Center at Dallas
Dallas, TX 75235

Thomas Held
Lister Hill National Center for Biomedical Communications
National Library of Medicine

GRAPHICS AND COMPUTER-BASED LEARNING

Chaired by Eugene Herman

Arthur Luehrmann
Kristina Hooper
Eugene A. Herman

ABSTRACT: Microcomputer Graphics and ANSI Standard BASIC

Arthur Luehrmann, Computer Literacy,
1466 Grizzly Peak Blvd., Berkeley,
CA 94708

Microcomputer graphics gives authors of educational software marvelous new ways of representing information and communicating ideas. But in terms of language, microcomputer graphics is a hodge-podge of untranslatable, exceedingly primitive statements far removed from the actual graphic problems faced by program authors. Each hardware manufacturer has created without much evident thought, its own set of Basic enhancements to exploit its own unique graphic capabilities. The result is students who learn graphics on one computer cannot use what they have learned on another computer. Committee X3J2 of the American National Standards Institute is near completion of its work in formulating a new ANSI standard for Basic. The new standard includes a full syntax for specifying high-resolution, two-dimensional, multi-color drawings, using a coordinate system appropriate to the problem at hand and independent of the particular hardware. The talk will describe the current status of graphics in Basic and will report on the present draft of ANSI Basic.

ABSTRACT: Pictorial Conversations--The Use of Computer Graphics to Enhance Understanding

Kristina Hooper, Visual Geometry Project,
University of California at Santa Cruz,
Santa Cruz, CA 95064

Pictures are extremely effective in many instances for portraying complex information. Simple diagrams can often convey concepts that are difficult to express linguistically. Graphs can show relationships that are obscure when represented in other forms. Pictures which can be systematically changed over time provide even more information than static pictures in many instances; buildup of pictorial information can show patterns that are obscure in static pictures. Rotation of three-dimensional objects on a number of axes makes the properties of these objects evident. Pictures that change over time as a function of an individual's state of knowledge and state of questioning--pictorial conversations--can be even more communicative than other pictorial forms. In these instances the observer is actively communicating in ways that are not typically available in the pictorial domain. Computer graphics offer an excellent opportunity for the interactive use of pictures in a communication framework.

ABSTRACT: Language-Independent Graphics for Education

Eugene A. Herman, Department of Mathematics,
Grinnell College, Grinnell, IA 50112

The process of creating and transforming pictures by computer is simple to understand. This assurance may be of little comfort, however, to an educator facing today's bewildering variety of graphics terminals, stand-alone computers with graphics capability, commercial plotting packages, languages with imbedded graphics commands, and graphics languages. Fortunately, ANSI committee X3H3 on Computer Graphics is preparing a standard that promises to clarify this process by specifying a collection of graphics

subroutines that can be implemented on any graphics device in any general-purpose high-level language. Represented on the committee are almost the entire graphics industry and most of the leading groups of graphics users. Thus, the standard promises to be rich enough to support the most advanced hardware and the most demanding applications. In this session we will address the question, "How can we in education expect to benefit from this standards work?"

CONTINUING EDUCATION

Frank A. Settle
Michael Pleva
Sally J. Weiler
George Mozes
Lebert R. Alley

ABSTRACT: Scientific Instrumentation Information Network and Curriculum (SIINC)

Frank A. Settle, Department of Chemistry, Virginia Military Institute, Lexington, VA 24450

Michael Pleva, Department of Chemistry, Washington and Lee University, Lexington, VA 24450

A computer-based, telephone-accessible system containing descriptive information for users of chemical instrumentation is currently being developed. The objective of Project SIINC is to test the feasibility of such a system for a limited number of instrumental methods. Task forces are currently preparing materials for gas-liquid chromatography, gas chromatography/mass spectrometry, and atomic absorption spectroscopy. Modules are planned for liquid chromatography, inductively coupled plasma emission spectroscopy, and a problem-oriented module, priority pollutants. This final module will demonstrate the ability of the preceding modules to provide the information required to solve problems of water analysis.

The system should provide concise, well-organized discussions and references for many types of users. Practicing professionals in science and engineering, instructors of courses involving instrumentation, and students, both undergraduate and graduate, can use the system to educate themselves on the selected instrumental methods.

ABSTRACT: Computer-Based Training for the Banking Industry

Sally J. Weiler, Educational Development Division, Bank Administration Institute, Park Ridge, IL 60068

Bank Administration Institute is the research and educational organization for the nation's banking industry. BAI's educational mission is maintained primarily through seminars, courses, and conferences conducted across the nation. However, increasing travel costs, coupled with the problem of locating qualified banker instructors, have recently led to an emphasis on using alternative educational delivery systems.

Initial emphasis has been in computer-based courseware. Through the Plato system, BAI delivers courses in regulatory compliance to bank officers and other banking personnel who need to be familiar with various regulations. BAI also provides a monthly electronic newsletter called RegsNow, which gives complete information on regulatory developments that affect the banking industry. It is delivered through Boeing's Scholar/Teach 3 system and is accessed in each bank on any of a variety of computer terminals.

Because of the success of these initial efforts, BAI will place increasing emphasis on computer-based courses in the coming year. Areas where courses can be expected include accounting, taxation, auditing, and others. These courses will be delivered through both Scholar/Teach 3 and Plato, and the possibility of utilizing the growing number of in-bank microcomputers will be explored.

ABSTRACT: Simulations on the Apple Microcomputer - A Network Approach

George Mozes, Library & Media Resources, Michael Reese Hospital and Medical Center
29th Street and Ellis Avenue, Chicago, IL 60616

This is a presentation of the work done by the Educational Development Unit (EDU) of Michael Reese Hospital and Medical Center to develop and disseminate simulations--patient management problems (PMP)--to run on the Apple microcomputer. PMPs originally developed as written simulations have been programmed for the Apple using Applesoft (Basic) and Pilot languages.

We are presently involved in organizing a network to bring together institutions and individuals interested or capable of creating, using, and disseminating PMPs. The network will provide a collection of PMPs, workshops on how to create PMPs for computers, and an ongoing review and evaluation process for the items in the collection.

Participants will have a chance to go through a few PMPs on the Apple, to ask questions, and to comment on issues raised in this presentation.

ABSTRACT: Experience with a Microprocessor/ Minicomputer Course for Continuing Engineering Education

Lebert R. Alley, Texas Tech University,
Department of Industrial Engineering,
Box 4130, Lubbock, TX 79409

A microprocessor/minicomputer laboratory and instructional course have been developed in the Texas Tech University Industrial Engineering Department. This presentation describes the array of laboratory equipment used, the design of the instructional course, and the successes and problems experienced to date.

The presentation will include photographs and documentation of laboratory equipment and layout, as well as written descriptions of the course.

A SUCCESSFUL EXPERIENCE WITH PROGRAMMING LANGUAGES AT A LIBERAL ARTS COLLEGE

Linore Cleveland
Vassar College

INTRODUCTION

At Vassar College the Computer Science Studies Program was created to serve the entire academic community. Students may include computer science in a major only under the option of an independent program. No well-defined major in computer science currently exists. Thus, the program provides courses in computer science which will benefit students in their various disciplines.

Students have expressed quite an interest in computer science, and our response has been to fashion courses that will provide a maximum amount of information without setting up a sequence beset with prerequisites. The one semester course taught in programming languages is a case in point. The course is designed to provide students with a framework in which they can evaluate and master a programming language(s) for a specific need. The goal of the course is not to teach programming in various languages that currently exist, but rather to build a structure for viewing programming languages that will, we hope, be as useful for a language that has never previously been encountered as for one to which they have been exposed.

The prerequisites for the course are an introductory course in programming and either a course in data structure or permission of the instructor. Students who are mature in their outlook and did well in the introductory course do well in the programming languages course without fulfilling the data structures prerequisite. Such students do find, however, that they need to read introductory material concerning strings and lists to appreciate some of the structures a programming language might provide.

COURSE ORGANIZATION AND OBJECTIVES

For the programming languages course, the objectives are to provide a technical framework for mastering a foreign or little known programming language, and to develop, within the students, a critical basis for the evaluation, selection, and acceptance of a programming language for a particular task(s). A third, less important, objective is to introduce a variety of programming languages. The choice of languages covered is dependent on the textbook chosen to supplement the course. A set offering the opportunity to illustrate both current and coming technical features and some history and raison d'etre of languages might include Algol-60, Fortran-77, Lisp, Snobol4, and Pascal.

The basis of much of the material presented in the course is the text Programming Languages by Jean Sammet. Ms. Sammet's coverage of the non-technical, or function, characteristics of programming languages provides a sound base for classroom presentation and discussion. Students respond by thinking critically, but with a proper historical perspective about the use of language in interactions with computer hardware.

As the following outline indicates, the course is divided into sections taught sequentially. The interrelations among topics surface at a time when the student has sufficient background for exploration.

Topical Outline: Programming Languages course:

- I. Introduction to programming languages
  - A. History
  - B. Definitions

- C. Advantages and disadvantages
- D. Classification of languages
- E. Selection factors

II. Functional characteristics
- A. Properties of languages
- B. Language purpose
- C. Conversion and compatibility considerations
- D. Standardization activity
- E. Means of language definition

III. Technical characteristics
- A. Language building blocks
- B. Subunit characteristics
- C. Exploration of data types
- D. Executable statement types
- E. Language structures

IV. Study of specific languages
- A. Algol-60
- B. Fortran-77
- C. Lisp
- D. Snobol4
- E. Pascal

I. INTRODUCTION TO PROGRAMMING LANGUAGES

This section of the course stimulates student thinking about programming languages: the whats, whys, whens, and wheres. Approximately 10% of available classroom time is spent on this topic. The following outline illustrates in more detail the material covered:

A. History
- 1. machine language
- 2. early languages
    - a. Short Code
    - b. Speedcoding System
    - c. Laning and Zierler System
    - d. A-2 and A-3
- 3. higher level languages
    - a. Fortran
    - b. Math-matic
    - c. Algol 58
    - d. Cobol

B. Definitions
- 1. programming language
    - a. "a language used to prepare computer programs"
    - b. definition by Sammet
- 2. source program
- 3. object program
- 4. compiler
- 5. interpreter

C. Advantages and disadvantages
- 1. advantages
    - a. ease of learning
    - b. ease of coding and understanding
    - c. ease of debugging
    - d. ease of maintaining and documenting
    - e. ease of conversion
    - f. reduction of elapsed time for problem solving
- 2. disadvantages
    - a. advantage illusionary
    - b. time required for compilation
    - c. inefficient object code
    - d. debugging difficulties
    - e. inability to utilize needed operations

D. Classification of languages
- 1. procedure oriented
- 2. non-procedural
- 3. problem oriented
- 4. application oriented
- 5. problem defining
- 6. problem solving
- 7. problem describing
- 8. reference
- 9. publication
- 10. hardware

E. Selection factors
- 1. suitability for problem areas and users
- 2. availability on desired computer
- 3. history and evaluation of previous use
- 4. efficiency of language implementation
- 5. compatibility and growth potential
- 6. non-technical characteristics
- 7. technical characteristics

The purpose of exploring the history of languages from the early 1950s is to put into perspective the climate in which programming languages developed. Such coverage allows the students to see how innovative such developments were at that time. They should also be impressed with the negative responses such innovations generated.

Jean Sammet defines a programming language as "a set of characters with rules for combining them which have the following characteristics," (1) machine code knowledge unnecessary, (2) potential for conversion to other computers, (3) instruction explosion, and (4) problem oriented notation. Contrasting this definition with definitions of the ANSI ilk (see point B.1.a. in the outline) sparks students to think critically about the concept of a programming language and its functional and technical characteristics. This serves as a motivating force for much of the material covered during the remaining segments of the course.

The definitions of source and object programs should simply reinforce notions developed in the introductory course all students had previously taken. However, it is necessary to spend time defining a compiler and interpreter. One should try to establish the basic difference between them, emphasizing the differing response to an input of a programming language instance.

Class discussion is a most successful method of approaching the topic of advantages and disadvantages. A list of advantages and disadvantages should be developed by the students. As each is presented, a rationale, justification, and rebuttal of the point should be discussed. Once prodded a bit, students do well at creating and defending such a list.

The various methods of classifying programming languages can be presented at this time and then referred to later as needed in discussing functional and technical characteristics. Similarly, selection factors should be outlined at this stage with only an introductory level of detail. They, too, will be referenced as course material develops.

II. FUNCTIONAL CHARACTERISTICS

In looking at functional characteristics of a programming language, one is exploring factors that affect the economic and political life of a programming language. It is important that students recognize that these considerations can be more important in the choice and acceptance of a programming language than the technical features offered. Covering this aspect occupies approximately 15% of the total course time. Specific topics covered are:

- A. Properties of languages
  - 1. generality and simplicity
  - 2. naturalness versus succinctness
  - 3. notational properties
  - 4. consistency
  - 5. efficiency
  - 6. ease of reading and writing
  - 7. error-proneness
  - 8. ease of learning
  - 9. self-documenting

- B. Language purpose
  - 1. type of application
  - 2. type of language
  - 3. type of potential user

- C. Conversion and compatibility considerations
  - 1. types of compatibility
    - a. machine independence
    - b. compiler independence
    - c. dialects and l-like languages
    - d. subsets and extensions
  - 2. relation to language definition
    - a. syntax
    - b. semantics
    - c. pragmatics
  - 3. conversion considerations
    - a. compatibility
    - b. Sift possibilities
    - c. translation

- D. Standardization activity
  - 1. purpose of standardization
  - 2. problems to be faced
    - a. when standardize?
    - b. technical concerns
      - technique to express definition
      - adherence to standard
      - upgrading of standard
      - subsetting
    - c. procedures for standardization
  - 3. current standardization activities (programming langs.)
    - a. general procedure
    - b. specific activities

- E. Means of language definition
  - 1. administrative concerns
  - 2. technical concerns
    - a. syntax, semantics, pragmatics
    - b. formalized notations
      - BNF as a metalanguage example
      - notations that include semantic concerns
        - W-Grammars
        - Production Systems
        - Vienna Definition Language
        - Attribute Grammars

In covering the properties of programming languages, most of the terms are self-explanatory. They can be introduced with an example, and students can contribute additional examples or experiences that relate to these properties. War stories serve beautifully to illustrate the importance of these properties in advancing the acceptance or rejection of a programming language. Students, even in their limited experience, usually have or have shared such observations with their peers.

Students should know that a language is not designed in a vacuum, but rather within the context of a specific application area(s), language type, and user ability. The intent of the section "language purpose" is to make this sufficiently clear.

Compatibility and conversion considerations have always been important in choosing a language, but in the environment of the 1980s, when technological advances result in more rapid expansion or alteration of computer hardware such considerations may be crucial. Compatibility and conversion cover a range of options which can be pointed out in this section. Students should be provided with sufficient information, including the definition of terms so that they can intelligently evaluate alternatives for a language choice, given the goals of an enterprise.

Students have probably heard some mention of standardization activity. It may have taken the form of a reference to Standard Fortran or to the multiplicity of Basics. However, students are not familiar with the relationship of standardization to the development of programming languages, nor with the activities involved in creating a standard. Such factors are explored in this unit in the course. Although students may still have questions about the standardization activity, they will have sufficient information to begin individual research to provide the answers.

The final topic covered in this section on functional characteristics is that of language definition. The student must be comfortable with the metalanguage concept and adept at reading a language definition using a metalanguage such as Backus Naur Form. The limitations of a language such as BNF and the current efforts to overcome such limitations should also be pointed out. The paper, "A Sampler of Formal Definitions" by M. Marcotty, H. F. Ledgard, and G. V. Bochmann, is a fine source document for discussion in this area.[2] The definition in the paper of the small language, Asple, can be used as an exercise base for reading a syntactical description in BNF, and as an introduction to other formal definition systems. A brief description of such a formal system is all students need absorb. However, additional time can be spent on the W-Grammar, working through a rather simple example so that the flavor of the approach can be visualized. The more advanced students in the course find this valuable.

III. TECHNICAL CHARACTERISTICS

In describing any language, in attempting to learn such a language, and in teaching a programming language, some type of structure is needed on which to hang the language features. The purpose of dealing with technical characteristics is to provide a generalized structure that can be used in identifying and organizing the features of a language. The structure developed should be sufficiently broad so that it fits any number of languages and can be used for future languages as well. The structure, as indicated in the following outline, has proved excellent for this purpose; it can be presented in the order given in the outline. If, as each point is covered, examples from a language or languages with which students are familiar can be given, structural pieces can be cemented in place. Reinforcement of the ideas, both singularly and as whole, is essential to ensure a meaningful use of this structure after students complete the course. Exercises and perhaps a paper on some programming language of the student's choice (with reference to the structure) are examples of such reinforcement. To do justice to this method of approaching language structure, approximately 40% of the time available during the semester is required.

Coverage of technical characteristics includes:

A. Building blocks
  1. character set
  2. tokens
    a. system defined - graphic operators, keywords, punctuation characters
    b. user defined - identifiers, literals, constants
    c. special points
      - reserved words
      - identifiers for program units
      - identifiers for data aggregates
        - number of subscripts
        - expression of subscripts
        - range of subscripts
        - subscripting subscripts
        - qualification of reference
      - noise words
      - punctuation relevance
  3. subunits
    a. non-executable: declarations, compiler directives
    b. smallest executable units
    c. sets of executable units

    d. loops
    e. subprograms: functions, subroutines, procedures
    f. complete program
B. Subunit characteristics
  1. iteration and recursion
    a. definition
    b. structure
    c. mechanisms
  2. delimiting subunits
    a. punctuation
    b. syntax
    c. special symbols
  3. scope of definitions
    a. global and local concepts
    b. defining names
      - explicit, contextual, implicit
    c. considerations of environment
  4. parameter passing
    a. formal parameters and arguments
    b. call by
      - value
      - reference
      - name

C. Exploration of data types
  1. unstructured attributes
    a. arithmetic
    b. logical or boolean
    c. character
    d. label
    e. pointer
    f. user defined
  2. structure attributes
    a. string
    b. array
    c. structure or record
    d. set or union
    e. file
  3. storage classes
    a. static
    b. automatic
    c. controlled
    d. based
  4. type checking

D. Executable statement types
  1. assignment statement
    a. intermingling rules for expressions
    b. locator expression
    c. value expression
  2. sequence control and decision making
    a. unconditional transfer
    b. controlled transfer
    c. alternative selection: if-then-else, case
  3. interaction with operating environment
    a. input/output statements
    b. debugging statements
    c. storage allocation statements
    d. concurrency coordination statements

E. Language structures
  1. sets of smallest executable units
    a. group
    b. block
    c. procedure
  2. loops
    a. range
    b. termination criteria
    c. parameters
    d. exit point
    e. common language forms
  3. subprograms
    a. arguments allowable
    b. recursion
    c. local/global variable definition
    d. return values
    e. internal/external

The first topic to be covered in this section is "building blocks." Here one tries to lay the foundation for further exploration. The notion of a token is valuable. In studying any language, to proceed any distance without clarifying what the rules for token formation are, what types of tokens exist in the language, and what freedoms the programmer has in forming and using tokens is difficult. Students taking the course have only been exposed to one language, so that notions such as reserved words, noise words, punctuation, and identifiers for data aggregates may well be encountered for the first time. This is the only place in this section of the course where character set and tokens are covered, so a thorough coverage is recommended. The material on subunits as building blocks will be explored further, with each subunit discussed in depth; its presence is to form a basis for discussions to follow.

The topic "subunit characteristics" is, perhaps, one of the most important to be covered. This may be the first time students have encountered these concepts, or even recognized such concepts are legitimate objects of consideration and can be divorced from specification of a particular language. Of the total time devoted to the unit on technical considerations, 35% to 45% should be expended in this area. The concepts are best appreciated and assimilated by using many examples and numerous exercises, especially for recursive concepts. Distinguishing between recursion mechanisms and recursion concepts is frequently a difficult task for students. Final examples combining recursion, iteration , scope of definition, and parameter passing can serve to solidify understanding of this topic.

Exploring data types is an exciting endeavor. Many of the unstructured and structured attributes are new to students, as is the world of storage classes. But, enthusiasm, discussion, and illustrations help ensure understanding. The storage class scheme as specified in the outline **is that** used by PL/I. The terminology of storage class is tremendously inconsistent; it reaches the extent that terms are used with different meaning in various languages. Students need to be aware of these contradictions and the need, when evaluating a particular language, to associate each term with the proper scheme.

Covering statement types should be extremely cut and dried. It is not! Any number of students have used statements of the various types without thinking through what the statement meant. This is especially true with the assignment statement, so time spent on the concepts of locator and value expressions is valuable. Many students have also never thought of subprogram invocations, return statements, premature loop exit statements and such, as sequence control statements. And few students have previously encountered statements of the "case" type. Those statements that interact with the operating environment are also infused with a sense of magic which a good discussion can help dispel. In addition, the interdependence of language and operating environment can be pointed out. The alternative, when the operating environment is deficient, is to create the environment within the object program. This is time consuming (both in terms of compilation and execution) and, perhaps, unrealistic in terms of the abstract machine the language is intended to mirror.

The final segment of this section of the course deals with language structures; that is, combining of units covered previously in ways that ultimately allow expression of a problem solution. This material flows easily and seems to be readily understood by the student.

IV. STUDY OF SPECIFIC LANGUAGES

The final 30% of the course is devoted to a study of specific languages. The coverage of each language should include some history, a description of the basic structures, and highlights of features unique or handled with particular skill in the language. A set of examples that illustrate the foregoing points in a way that emphasizes the strengths and weaknesses of the language should be used in the presentation. For each language covered, students should be given a small program written in the language. They should be asked to determine the purpose of the program by investigating the expressed solution. If facilities allow students to actually write programs in the language (currently not possible at Vassar College), a reasonable approach would be to have them modify existing programs. This allows a sampling of the language flavor without worrying unnecessarily about the details of writing programs in the language.

CONCLUSIONS

This course was taught as a one semester (13 weeks at 150 minutes per week lecture) course at Vassar College in the fall of 1978 and again in the fall of 1980. Although it was created prior to the Cirriculum '78 ACM specifications, it addressed many of the topics in course CS 8 (Organization of Programming Languages). Favorable student reaction indicated that the course had definitely achieved its objectives-so did work students submitted during and at the completion of the semester. From an instructor's viewpoint, the course is exciting to teach. Much of the material is either new to students or suggests a new dimension to previously encountered material.

REFERENCES:

1. Sammet, Jean E., Programming Languages: History and Fundamentals, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1969.

2. Marcotty, M., H.F. Ledgard, G.V. Bochmann, "A Sampler of Formal Definitions", ACM Computing Surveys, Volume 8, Number 2, June 1976, pp191-276.

3. Austing, Richard H., Bruce H. Barnes, Della T. Bonnette, Gerald L. Engel, Gordon Stokes, "Curriculum '78", Communications of the ACM, Volume 22, Number 3, March 1979, pp 147-166.

4. Organick, E.I., A.I. Forsyth, R.P. Plummer, Programming Language Structures, Academic Press, New York, 1978.

5. Tucker, Allen B., Jr., Programming Languages, McGraw-Hill Book Company, New York, 1977.

6. Pratt, T.W., Programming Languages: Design and Implementation, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1975.

7. Elson, Mark, Concepts of Programming Languages, Science Research Associates, Inc., Chicago, IL, 1973.

8. Wasserman, Anthony I., Tutorial: Programming Language Design, The Institute of Electrical and Electronic Engineers, Inc., New York, 1980.

9. Wexelblat, R.L., Ed., "Preprints, ACM SIGPLAN History of Programming Languages Conference", SIGPLAN NOTICES, Volume 13, Number 8, August 1978.

THE C-BASIC INTERPRETER:
A LEARNING TOOL for COMPILER CONSTRUCTION

by
John A. Rohr
Computer Science Department
California State Polytechnic University, Pomona
Pomona, California

ABSTRACT

The C-Basic (Class - Basic) language and interpreter is designed to teach students practical techniques to implement compilers and interpreters. The language is a subset of the Basic* language. In a one-quarter course on the theory and design of compilers and interpreters, students construct a scanner, a parser, an intermediate code generator, and an interpreter for C-Basic. This paper describes the language and the project used in the course. Several appendices are included, giving specific details about several aspects of the project.

INTRODUCTION

The Computer Science Department at California State Polytechnic University in Pomona, California, teaches a sequence of courses in the theory and design of translators. The first course presents assemblers and loaders; the second, compilers and interpreters. The student project for the second course is an interpreter for C-Basic (Class - Basic). The following sections of this paper present the language specification, the project description, observations about the project, and a set of appendices giving specific details about several aspects of the project.

LANGUAGE SPECIFICATION

The C-Basic language is a subset of the Basic language. A single extension is made to allow variable names (symbols) of up to six characters. The character set used consists of the upper-case letters, the digits 0-9, and the special characters +, -, *, /, ↑, =, >, <, (,), comma, space, and end-of-line. The lexical items (tokens) generated by the scanner can be divided into six categories: punctuation, operators, keywords, line numbers, constants, and identifiers. Keywords are reserved words: they may not be used as identifiers. The C-Basic lexical items are listed in Appendix A.

A C-Basic program consists of any number of non-end statements terminated by an end statement. Ten different types of statements are provided plus the end statement. Positive or negative integer and decimal constants may consist of up to five digits plus an optional sign. One-dimensional arrays are provided. Arithmetic and Boolean expressions are also incorporated. Although the language is simple, it does provide enough capability to actually write and run simple computational programs. The complete C-Basic syntax is given in Appendix B.

INTERPRETER IMPLEMENTATION

The C-Basic Interpreter is implemented by each student in the class either alone or with one partner. The task of writing the interpreter is partitioned into five phases. Each phase provides an essential portion of the interpreter. The five phases of the project are the:

1) symbol table
2) scanner
3) parser
4) code generator
5) interpreter

Symbol Table

The task of writing a set of symbol table routines is assigned on the first day of class. We assume that students know how to implement symbol tables from their work in the assembler class. By assigning the symbol-table phase on the first day of class, students can begin immediately on their projects. The only information they require are the length of the symbols (up to six characters), the valid characters for symbols (letters and digits only), the number of symbols to be supported (64), and the fields of the symbol table (symbol, type, location, dimension). The students are also told that they may need to retrieve, change, or augment symbol data after it has initially entered. The students are allowed to chose their own symbol table algorithms. A sample symbol table is given in Appendix I.

---

*Basic (Beginner's All-purpose Symbolic Instruction Code) is a registered trademark of Dartmouth College.

Scanner

The scanner or lexical analyzer is implemented as the second phase of the project. The scanner recognizes six categories of lexical items or tokens. Punctuation marks, operators, and keywords are each assigned unique token codes. Line numbers, constants, and identifiers are assigned one token code per category, and an index number is returned with the token code to uniquely identify the specific token.

Keywords are entered into the symbol table during the initialization of the interpreter, which simplifies identifying keyword tokens and prevents using the keywords as identifiers. Each line number, constant, and identifier is entered into the symbol table with default values when first encountered.

The students chose their own method of implementing the scanner. Two methods are discussed in class. Transition diagrams implemented by short code segments for each state are the simplest to understand and implement. Deterministic finite automata are discussed as an alternate method. The use of automatic scanner-generators is also discussed, but this method is not permitted for student projects.

Parser

The parser or syntax analyzer, which is the third phase of the project, is the most complex phase. A predictive parser is implemented. This is a table-driven, top-down, recursive-descent parser. There are actually four sub-phases in this phase:

1) eliminating left recursion followed by left-factoring
2) generating FIRST and FOLLOW
3) generating the parsing table
4) coding

The first three are each given as homework assignments. Left recursion must first be eliminated from the grammar then it must be left-factored to be in a form suitable for a recursive-descent parser. The set FIRST (α), which is the set of all possible first terminals for α in the production $\beta \rightarrow \alpha$, must be determined for each production. Also, the set FOLLOW (A), which is the set of terminals which can immediately follow the nonterminal A in some sentential form, must be determined for each nonterminal of the grammar. Then the parsing table must be constructed using the FIRST and FOLLOW sets.

After the students have generated their own parsing tables, they are given a table to use in their projects, assuring uniformity in parsing. The parsing algorithm is given in the textbook[1] and additional reference material is available.[2] The code for this phase is short, but a rather sizeable data table (the parsing table) must be generated. The C-Basic syntax (with semantics) used for this phase is given in Appendix C. The parsing table is given in Appendix F.

Code Generator

The code generator, which is the fourth phase of the project, is implemented as a simple extension to the parser. First, the syntax is augmented with semantic actions. Then the parsing algorithm is augmented to include performing semantic actions. There are two types of semantic actions: one type emits an operator code; the other type performs a specialized action. There are 22 C-Basic operators and 8 special semantic actions. The augmented parsing algorithm which is used adds checks for semantic actions in the production and performs the actions indicated as they are encountered in the productions.

The code which is emitted by the code generator is Polish-postfix code which would be the intermediate code for a compiler. The Polish-postfix code can be used directly by the interpreter. The emitted code contains only operators and operands. Operands are emitted as positive numbers which are the symbol table indices. Operators are emitted as negative numbers which identify the operators. The C-Basic syntax with semantics is given in Appendix C. The C-Basic operators are listed in Appendix D. A sample of C-Basic Polish-postfix code is given in Appendix H.

Interpreter

The interpreter, the fifth phase of the project, provides a virtual machine to execute the Polish-postfix code. The interpreter uses an evaluation stack to store operands as they are encountered in the code and a set of subroutines to perform the operations. There is a unique subroutine for each operator. Each subroutine obtains its arguments (if any) from the evaluation stack and returns its result (if any) to the evaluation stack. Local storage may be used in some of the operator routines for data such as subroutine return points, etc. The source code, postfix code, and symbol table for a test case are shown in Appendices G, H, and I respectively.

PROJECT EVALUATION

Although the primary purpose of the C-Basic student project is to teach the student compiler-writing techniques, evaluation of work is necessary to determine course grades. Reports are required for each phase and for the entire project. The reports consist of a description of the phase, a narrative about the solution, a specification of the algorithm(s) used by a flowchart or design-language listing, the code written, and the execution output for the test case. The final report is a complete description of the project including the data base, files, algorithms, and operating instructions.

The entire project constitutes one-half of the class grade. (Examinations constitute the other half.) The final report constitutes half of the entire project grade. The other half is equally distributed among the five phase reports. The grades for the phase report and the final report are based on the correctness of the code and the effectiveness in communicating the design and implementation in the report.

OBSERVATIONS

The C-Basic interpreter is a valuable learning tool for students studying the theory and design of compilers and interpreters. The first four phases of the project are equally applicable to a compiler or an interpreter. For a one-quarter course, there is barely sufficient time to complete the project with an interpreter. (Sometimes the interpreter must be left as an optional phase which will be done only by the better students.) In a one-semester course it should be possible to include assembly--or machine--language code generation as well as some optimization and error recovery.

Finally, although there are scanner generators and parser generators available to automate much of the work involved in generating a compiler, it is imperative that students generate a compiler or interpreter once manually to understand and appreciate the automated methods available. The implementation of C-Basic is one way of accomplishing this.

REFERENCES

1. Aho, Alfred V. and Jeffrey D. Ullman, Principles of Compiler Design, Addison-Wesley Publishing Company, Reading, Massachusetts, 1977.

2. Lewis, Philip M., II, Daniel J. Rosenkrantz, and Richard E. Stevens, Compiler Design Theory, Addison-Wesley Publishing Company, Reading, Massachusetts.

APPENDICES

Appendix A

C-Basic Lexical Items

Punctuation

Code	Character
0	End-of-line
1	,
2	(
3	)

Operators

Code	Character
4	+
5	-
6	*
7	/
8	↑
9	=
10	>=
11	<=
12	>
13	<
14	<>

Keywords

Code	Character
15	LET
16	GOTO
17	ON
18	IF
19	THEN
20	FOR
21	TO
22	STEP
23	NEXT
24	DIM
25	GOSUB
26	RETURN
27	STOP
28	END

Special Categories

Code	Character
29	Line Number
30	Constant
31	Identifier

A line number consists of 1 to 5 decimal digits and is the first lexical item of a statement.

A constant consists of 1 to 5 decimal digits and is not the first lexical item of a statement.

An identifier is a symbol consisting of 1 to 6 letters and digits, the first of which must be a letter.

Appendix B

C-Basic Syntax

This syntax for C-Basic is in modified BNF where the notation {...}$^{j}_{i}$ indicates that the item inside the brackets must occur at least i times and no more than j times.

Production Number   Production

1. <program>:= {<statement>}$^{n}_{o}$ <terminal st>
2. <statement>:= <line no>{<let st>|<goto st>|<on st>|<if st>|<for st>|<next st>
   |<dim st>|<go sub st>|<return st>|<step st>}$^{1}_{o}$ <eol>
3. <terminal st>:= <line no> <end st> <eol>
4. <eol>:= EOL
5. <let st>:= ‸LET‸ <variable> = <expression>
6. <goto st>:= ‸GOTO‸ <line no>
7. <on st>:= ‸ON‸ <expression> ‸GOTO‸ <line no>{,<line no>}$^{n}_{o}$
8. <if st>:= ‸IF‸ <boolean exp> ‸THEN‸ <line no>
9. <for st>:= ‸FOR‸ <simple variable> = <expression> ‸TO‸ <expression>
   {‸STEP‸<expression>}$^{1}_{o}$
10. <next st>:= ‸NEXT‸ <simple variable>
11. <dim st>:= ‸DIM‸ <array dim>
12. <go sub st>:= ‸GOSUB‸ <line no>
13. <return st>:= ‸RETURN‸
14. <step st>:= ‸STOP‸
15. <end st>:= ‸END‸
16. <expression>:= <multiply factor>|<prefix op><expression>|<expression> {+|-}$^{1}_{1}$
    <multiply factor>
17. <multiply factor>:= <multiply factor>{*|/}$^{1}_{1}$ <involution factor>|<involution
    factor>
18. <prefix op>:= +|-
19. <involution factor>:= <term>|<term>↑<term>
20. <term>:= <constant>|<variable>|(<expression>)
21. <variable>:= <simple variable>|<subscripted variable>
22. <simple variable>:= <identifier>
23. <identifier>:= <letter>{<letter>|<digit>}$^{5}_{0}$
24. <subscripted variable>:= <identifier>(<expression>)
25. <constant>:= {<digit>}$^{5}_{1}$
26. <boolean exp>:= <expression><relation><expression>
27. <relation>:= =|>=|<=|>|<|<>
28. <array dim>:= <identifier>(<constant>)
29. <digit>:= 0|1|2|3|4|5|6|7|8|9
30. <letter>:= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

Appendix C
C-Basic Syntax (with Semantics)

Production Number	Production
1.	P' → P
2.	P → S·P
3.	TS
4.	S → ⑦ L·ES·EOL
5.	ES → LS
6.	GTS
7.	OS
8.	IS
9.	FS
10.	NS
11.	DS
12.	GSS
13.	RS
14.	SS
15.	ε
16.	TS → ⑦ L·ENS·EOL ⑧
17.	EOL → eol
18.	LS → let V = E $\boxed{=}$
19.	GTS → goto ① L $\boxed{\text{GOTO}}$
20.	OS → on E goto ① L OST ③ $\boxed{\text{ON}}$
21.	OST → , ① ④ L·OST
22.	ε
23.	IS → if BE then ① L $\boxed{\text{IF}}$
24.	FS → for SV = E to E·FST $\boxed{\text{FOR}}$
25.	FST → step E
26.	ε②
27.	NS → next SV $\boxed{\text{NEXT}}$
28.	DS → dim AD
29.	GSS → gosub ① L $\boxed{\text{GOSUB}}$
30.	RS → return $\boxed{\text{RETURN}}$
31.	SS → stop $\boxed{\text{STOP}}$
32.	ENS → end $\boxed{\text{STOP}}$
33.	E → MF·ET
34.	+E·ET
35.	-E $\boxed{\text{u-}}$ ET
36.	ET → +MF $\boxed{+}$ ET
37.	-MF $\boxed{-}$ ET
38.	ε
39.	MF → IF·MT
40.	MT → *IF $\boxed{*}$ MT
41.	/IF $\boxed{/}$ MT
42.	ε
43.	IF → T·IT
44.	IT → ↑T $\boxed{\uparrow}$
45.	ε
46.	T → ① C
47.	V
48.	(E)
49.	V → SV
50.	SBV
51.	SV → ① I
52.	SBV → ① I (E) $\boxed{\text{SUBSCRIPT}}$
53.	BE → E·BT
54.	BT → = E $\boxed{=_R}$
55.	>= E $\boxed{>=_R}$
56.	<= E $\boxed{<=_R}$
57.	> E $\boxed{>_R}$
58.	< E $\boxed{<_R}$
59.	<> E $\boxed{<>_R}$
60.	AD → ⑤ I (⑥ C)

Appendix D
C-Basic Operators

Opcode	Operator	First Operand	Second Operand	Other Operands
1.	+	A	B	
2.	-	A	B	
3.	- (Unary)	A		
4.	*	A	B	
5.	/	A	B	
6.	↑	A	B	
7.	$=_R$	A	B	
8.	$>=_R$	A	B	
9.	$<=_R$	A	B	
10.	$>_R$	A	B	
11.	$<_R$	A	B	
12.	$<>_R$	A	B	
13.	=	A	B	
14.	GOTO	L		
15.	ON	AE	L_1	L_2 $L_3 \ldots L_N$ N
16.	IF	BE	L	
17.	FOR	S	AE	AE AE
18.	NEXT	S		
19.	GOSUB	L		
20.	RETURN			
21.	STOP			
22.	SUBSCRIPT	S	E	

A: First Operand	L: Label
B: Second Operand	Li: Label
AE: Arithmetic Expression	N: Number
BE: Boolean Expression	S: Symbol

Appendix E
C-Basic Semantic Actions

1. Emit Operand: Symbol Table Index
2. Emit Literal 1: Symbol Table Index
3. Emit Literal (OPCNT+1): Symbol Table Index
4. OPCNT = OPCNT+1
5. SBVI = Table Index for I
6. Put Dimension in I.DIM in Symbol Table
7. Put LOCCTR in I.LOC in Symbol Table Set OPCNT = 0
8. End Compilation

Appendix F

C-BASIC Parsing Table

Index	Non-Terminal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	P'																														1		
2	P																														A		
3	S																														4		
4	ES	15															5	6	7	8		9			10	11	12	13	14				
5	TS																														16		
6	EOL	17																															
7	LS																18																
8	GTS																	19															
9	OS																		20														
10	OST	22	21																														
11	IS																			23													
12	FS																					24											
13	FST	26																						25									
14	NS																								27								
15	DS																									28							
16	GSS																										29						
17	RS																											30					
18	SS																												31				
19	ENS																													32			
20	E			33		34	35																									33	33
21	ET	38			38	36	37				38	38	38	38	38	38		38			38		38	38									
22	MF			39																												39	39
23	MT	42			42	42	42	40	41		42	42	42	42	42	42		42			42		42	42									
24	IF			43																												43	43
25	IT	45			45	45	45	45	45	44	45	45	45	45	45	45		45			45		45	45									
26	T			48																												46	47
27	V																																B
28	SV																																51
29	SBV																																52
30	BE			53		53	53																									53	53
31	BT										54	55	56	57	58	59																	
32	AD																																60

A: Use 3 if the next token is END; use 2 otherwise.

B: Use 50 if the next token is (; use 49 otherwise.

Appendix G

C-Basic Test Case - Source

```
00100  DIM ARRAY(100)
00200  LET A = 1
00300  LET BC = 2
00400  LET DEF = 3
00500  LET GHIJ = 4
00600  LET KLMNO = 5
00700  LET PQRSTU = 6
00800  FOR VWXYZ = 1 TO 100
00900  LET ARRAY(VWXYZ) = VWXYZ
01000  NEXT VWXYZ
01100  FOR I1 = 10 TO 100 STEP 10
01200  LET ARRAY (I1) = 10*ARRAY (I1)
01300  NEXT I1
01400  FOR K = 1 TO 3
01500  ON K GOTO 1600, 1700, 1800
01600  LET VALUE = ((A + BC)*DEF/GHIJ)
       ↑KLMNO + 2*PQRSTU - 36
01700  IF K <= 1 THEN 1900
01800  LET VALUE = - (VALUE↑2)
01900  NEXT K
02000  GOSUB 2200
02100  GOTO 02400
02200  LET VALUE = -VALUE
02300  RETURN
02400  STOP
02500  END
```

Appendix H

C-Basic Test Case - Postfix Code

	Location	Numeric	Mnemonic
Line 200	0	5	A
	1	6	1
	2	-13	$\boxed{=}$
Line 300	3	8	BC
	4	9	2
	5	-13	$\boxed{=}$
Line 400	6	11	DEF
	7	12	3
	8	-13	$\boxed{=}$
Line 500	9	14	GHIJ
	10	15	4
	11	-13	$\boxed{=}$
Line 600	12	17	KLMNO
	13	18	5
	14	-13	$\boxed{=}$
Line 700	15	20	PQRSTU
	16	21	6
	17	-13	$\boxed{=}$
Line 800	18	23	VWXYZ
	19	6	1
	20	3	100
	21	6	1
	22	-17	$\boxed{\text{FOR}}$
Line 900	23	2	ARRAY
	24	23	VWXYZ
	25	-22	$\boxed{\text{SUBSCRIPT}}$
	26	23	VWXYZ
	27	-13	$\boxed{=}$
Line 1000	28	23	VWXYZ
	29	-18	$\boxed{\text{NEXT}}$
Line 1100	30	27	I1
	31	28	10
	32	3	100
	33	28	10
	34	-17	$\boxed{\text{FOR}}$
Line 1200	35	2	ARRAY
	36	27	I1
	37	-22	$\boxed{\text{SUBSCRIPT}}$
	38	28	10
	39	2	ARRAY
	40	27	I1
	41	-22	$\boxed{\text{SUBSCRIPT}}$
	42	-4	$\boxed{*}$
	43	-13	$\boxed{=}$
Line 1300	44	27	I1
	45	-18	$\boxed{\text{NEXT}}$
Line 1400	46	32	K
	47	6	1
	48	12	3
	49	6	1
	50	-17	$\boxed{\text{FOR}}$
Line 1500	51	32	K
	52	34	#1600
	53	35	#1700

	Location	Numeric	Mnemonic
	54	36	#1800
	55	12	3
	56	-15	$\boxed{\text{ON}}$
Line 1600	57	37	VALUE
	58	5	A
	59	8	BC
	60	-1	$\boxed{+}$
	61	11	DEF
	62	-4	$\boxed{*}$
	63	14	GHIJ
	64	-5	$\boxed{/}$
	65	17	KLMNO
	66	-6	$\boxed{\uparrow}$
	67	9	2
	68	20	PQRSTU
	69	-4	$\boxed{*}$
	70	-1	$\boxed{+}$
	71	38	36
	72	-2	$\boxed{-}$
	73	-13	$\boxed{=}$
Line 1700	74	32	K
	75	6	1
	76	-9	$\boxed{\text{<=R}}$
	77	39	#1900
	78	-16	$\boxed{\text{IF}}$
Line 1800	79	37	VALUE
	80	37	VALUE
	81	9	2
	82	-6	$\boxed{\uparrow}$
	83	-3	$\boxed{\text{U-}}$
	84	-13	$\boxed{=}$
Line 1900	85	32	K
	86	-18	$\boxed{\text{NEXT}}$
Line 2000	87	41	#2200
	88	-19	$\boxed{\text{GOSUB}}$
Line 2100	89	43	#2400
	90	-14	$\boxed{\text{GOTO}}$
Line 2200	91	37	VALUE
	92	37	VALUE
	93	-3	$\boxed{\text{U-}}$
	94	-13	$\boxed{=}$
Line 2300	95	-20	$\boxed{\text{RETURN}}$
Line 2400	96	-21	$\boxed{\text{STOP}}$
Line 2500	97	-21	$\boxed{\text{STOP}}$

Appendix I

C-Basic Test Case - Symbol Table

Index	Symbol	Type	Location	Dimension
1	#100	24	0	0
2	ARRAY	26	14	100
3	100	25	13	0
4	#200	24	0	3
5	A	26	0	0
6	1	25	1	0
7	#300	24	3	3
8	BC	26	2	0
9	2	25	3	0
10	#400	24	6	3
11	DEF	26	4	0
12	3	25	5	0
13	#500	24	9	3
14	GHIJ	26	6	0
15	4	25	7	0
16	#600	24	12	3
17	KLMNO	26	8	0
18	5	25	9	0
19	#700	24	15	3
20	PQRSTU	26	10	0
21	6	25	11	0
22	#800	24	18	5
23	VWXYZ	26	12	0
24	#900	24	23	5
25	#1000	24	28	2
26	#1100	24	30	5
27	I1	26	114	0
28	10	25	115	0
29	#1200	24	35	9
30	#1300	24	44	2
31	#1400	24	46	5
32	K	26	116	0
33	#1500	24	51	6
34	#1600	24	57	17
35	#1700	24	74	5
36	#1800	24	79	6
37	VALUE	26	117	0
38	36	25	118	0
39	#1900	24	85	2
40	#2000	24	87	2
41	#2200	24	91	4
42	#2100	24	89	2
43	#2400	24	96	1
44	#2300	24	95	1
45	#2500	24	97	1

TEACHING FUNDAMENTAL COMPUTER TECHNOLOGY BY SUBJECT MATTER PROFESSIONALS USING PROGRAMMABLE CALCULATORS

Reed D. Taylor and Michael W. Woolverton

Department of Agricultural Economics and Rural Sociology
Ohio Agricultural Research and Development Center

ABSTRACT

This paper describes an approach to teaching basic data processing techniques in specific subjects using programmable calculators. The Texas Instruments Model 59 programmable calculator provides undergraduate students with an extremely low-cost means of gaining hands-on experience with computer hardware and software. A survey of students taking the course indicated they felt the effort expended was high relative to other courses, but were highly satisfied with the course, with the availability of the miniature computers, and with the perceived future personal utility of the course. The authors conclude that teaching fundamental computing techniques with programmable calculators may be more effective than requiring students to receive instruction based on large, mainframe computers in computer science departments.

I. INTRODUCTION

In 1972, Charles E. French, then Chairman of the Department of Agriculural Economics at Purdue University stated:

> Historians may not fully concur, but certain engulfing urges seem to sweep through our economic and social system from time to time. These become societal prime movers. Some such urges can be identified, e.g., the urge to explore, the urge to mechanize, the urge to organize, the urge to accelerate, and the urge to socialize. Two or more of these may interface at times. Cybernetics seems to be such a prime mover balanced on modern urges to industrialize and accelerate. In this sense it is profound and powerful among events of our time and has influenced agriculture (society) positively. (French, 1972)

Rapid development of data processing equipment and procedures over the past decade has placed the power of computers into the hands of anyone. We have become the computerized society.

This pervasive impact of cybernetics on society presents a unique challenge to the educational community. In fact, the National Science Foundation stresses that due to the high potential for influencing the nation positively, teaching basic computer principles and concepts to students with scientific majors should receive high priority for use of educational resources over the next decade (National Science Foundation, 1979). The Foundation concludes that developing appropriate educational programs is not only desirable but inevitable, and that federal funds and influence should be expended to accelerate the process. While the National Science Foundation stresses the need for educating students with scientific majors in computer technology, the same needs can be implied for students in other fields of study.

II. STATEMENT OF THE PROBLEM

The most pressing problem facing the educational community is to prepare students to cope with a world of diminishing natural resources wherein the costs of extracting those resources, with any given technical base, continues to increase. Cybernetic technology seems to be a basic ingredient of practically all emerging technologies that might have the combined capacity of maintaining or improving standards of living. We also believe that educators in practically every field of study from the sciences to the liberal arts improve the quality of education in their field by greater use of presently available cybernetic technology, which can usually be done at very low cost.

Traditionally, educators have turned to computer science departments to train students in data processing. Computer science departments usually use large, expensive, mainframe equipment. Students normally prepare punched cards for batch processing or use terminals for time-sharing. Such programs emphasize training students to become programmers and/or computer scientists. We believe that agriculture students and others tend to shun such courses. Thus, the percentage of students with fundamental data processing training graduating from colleges remains low.

While the above approach was probably valid when computer equipment was expensive, sensitive, and difficult to use, now every student should gain a basic concept of modern data processing technology, and these concepts can be most effectively taught by subject matter professionals. Computer equipment has been miniaturized, made easier to use, become less expensive, and improved in reliability. Having subject matter professionals teach data processing reserves computer science departments and their large, mainframe computers for training computer scientists further educating other students desiring advanced data processing techniques, and for researching methods of data processing. This shift could educate a greater number of students more effectively in methods of incorporating cybernetic technology into their various subjects, thus enabling them to become more effective problem solvers.

III. OBJECTIVES OF THE PAPER

Objectives of the paper are to present:

1. a brief description of programmable calculator technology and the potential for using it to educate students on fundamental computer technology.

2. a description of a course designed and taught by the authors to teach basic computer technology and applications to agricultural students.

3. an evaluation of the course as to its effectiveness in meeting course objectives.

4. implications for professionals in other subject matter areas.

IV. THE TECHNOLOGY

A. Historical Perspective

Present computers operate thousands of times faster than the first models and at a fraction of the cost per unit of work accomplished. They are also easier to access; more accurate and reliable; simpler to program; less sensitive to environmental conditions; and occupy less space per unit of capacity.

Two developments of special significance to small users have been time-sharing and the advanced programmable calculator. Time-sharing, in its broadest sense, allows one almost instant access to computer programs and data banks at any location from any location. All that is needed is a terminal, a telephone, electrical current, and a contract with the appropriate computer center (Taylor, 1976). This development of the late 1960s and early 1970s was especially significant to educators and small businessmen since it gave them access to computer technology at dispersed locations at relatively low cost.

Battery powered, hand-held calculators became available in the early 1970s. Technology has been advanced rapidly so that now some programmable calculators are sophisticated computers. Even more startling technological improvements lie ahead. Experts predict VLSI (Very Large-Scale Integration) will make it possible, perhaps within five years, to compress the number-handling proficiency of a present day, large computer into a single part about the size of a match head. Such superchips could prove to be the technological equivalent of the leap from transistors to integrated circuits in the early 1960s (Shaffer, 1979).

Even now many programs previously available to the small user only through time-sharing can be adapted for use on programmable calculators. These units are particularly attractive because they provide access to computer capability cheaply. An advanced programmable calculator can be purchased for about \$200, a compatible printer for about \$150, and exchangeable modules of programs for about \$35 each, for a total outlay of less than \$500. Operation costs are negligible.

A. Characteristics of Advanced Programmable Calculators Enabling Students to Meet Educational Objectives.

Advanced programmable calculators promise to help meet a multitude of educational objectives. New models may be programmed by the user and allow preserving programs and data on read/write media. have solid state exchangeable program modules, allow upward compatibility in both learning and new hardware introduction, are inexpensive and portable, and offer versatility. The unit discussed in this paper as an example of programmable calculators is the Texas Instruments Model 59. Models with similar capabilities are available from other manufacturers.

Programmability is the most attractive feature of these units. Fairly sophisticated programs can be developed and executed. Because the units can be programmed, students using a TI-59 or similar programmable calculator can write, code, and execute programs. The graphic representation of the problem to be solved, i.e., the flowchart, takes on new meaning when program steps and data are entered into the unit. Execution provides immediate feedback to students. Applications requiring numerous calculations can be handled with little difficulty. However, the small memory compared to more sophisticated computer systems does place an upward limit on the number of programming steps and the amount of data that can be entered.

Late model programmable calculators allow users to save programs and the contents of memory registers on read/write media, magnetic cards in the case of the TI-59. So, users can save for future use an unlimited number of programs. To solve a particular problem, a user loads a program into the hand computer simply by having the unit read magnetic cards. Data may be entered by hand or from magnetic cards.

Commonly used programs are often available on modules, each containing about 25 programs. These programs or portions of them can be used individually in problem solving and can also be used as subroutines in individually written programs. This subroutine capability greatly increases the potential scope of user-written programs.

Concepts and skills learned and developed by students in working with hand computers ease the transition to minicomputers, time-sharing, and mainframe units. For example, in using the programmable calculator, students must assign locations for both instructions and data and make provisions for recall from assigned locations. This requirement, which in a large computer is automatically handled by a compiler, makes students more aware of basic computer operations.

Manufacturers intend to maintain upward compatibility in new hand-held computer models of the future. The new units will include many features of present models, for example key layout. Students becoming proficient in the use of present models should find little difficulty in upgrading to newer, more powerful equipment.

Educators are constantly reminded of budget limitations. Compared to other automatic data processing equipment, the cost of providing hands-on experience with programmable calculators is very low. Cost comparisons will be made in a later section.

Providing students with what amounts to their own personal computer adds greatly to their interest in and satisfaction with a basic computer education course. Because the units are small they can be used in different classrooms and even can be checked out of the library. For example, the TI-59 weighs less than a pound and can operate for about three hours on a battery charge.

Versatility of programmable calculators also adds satisfaction by enabling students to acquire knowledge from a preexisting base since nearly all have used simple hand-held calculators. A programmable calculator can be used by students simple as a calculator or as a computer using internal programs contained in exchangeable modules, programs developed by others which are entered by coding, and programs written by students themselves. This progression from use as a calculator to self-programming can ease student transition into the sometimes intimidating world of computers.

An important factor in any computer system is availability of appropriate programs to solve problems. Fortunately, development of programmable calculator programs has been rapid. Many programs written for other computer systems have been adapted to the compact units.

Solid state exchangeable library modules presently available on the TI-59, for example, include: master, agriculture, applied statistics, real estate and investment, aviation, marine navigation, surveying, leisure, securities analysis, and busines decisions. Additional library modules are being developed as demand is sufficient to justify costs.

In addition to the modules, Texas Instruments offers specialty packets which are collections of programs designed for special interest groups. The user simply keys the desired program into the unit. Finally, one can join user groups wherein an individual may both contribute and receive programs.

V. THE COURSE 'COMPUTERS IN AGRICULTURAL DECISIONS'

A. Educational Objectives

The basic problem in the College of Agriculture and Home Economics at The Ohio State University was to provide students a working knowledge of applied computer techniques they could use in upper-level courses, graduate-level courses, and on the job. They did not need the depth and breadth of training a sequence of computer science courses might give them. But still they needed to know the elements of data handling in information systems; they needed to be acquainted with computer system components and understand the appropriate terminology; students also needed some knowledge of basic computer programming so they could understand computerized problem-solving approaches.

The course that evolved from this set of needs is "Computers in Agricultural Decisions" (Taylor, 1980). The overall objective of the course is to develop in students the ability to solve agricultural decision-making problems with the help of computer systems. The problem-solving approach is taught as a four-step sequence: 1) problem analysis; 2) flowchart application; 3) coding and executing the program; and 4) documentation.

B. Anatomy of the Course

Enrollment is limited to 86 students (room capacity) each quarter with student demand exceeding capacity most quarters. All students meet together three periods per week for lecture. The class is then divided into four groups, based on subject matter interest, for discussion/laboratory periods. Each group meets once a week for a two-hour period. Students are required to purchase two texts. The first, Introduction to the Computer: The Tool of Business, is used to introduce them to basic computer concepts (Fuori, 1981). The second, Programmable Calculators: Business Applications, is specific to the equipment being used (Aronofsky, 1978).

Experience has shown that an ideal complement of equipment for the course is 25 programmable calculators, 25 compatible printers, and 15 agricultural modules. This allows 10 units to be made available to students in libraries, learning aids laboratories, or other locations on a continuous basis; one calculator for each two students in discussion/laboratory sections; one unit for the instructor; and three spares to handle unusually large sections and to replace units being repaired. McGrann and Edwards, in their agricultural extension education experience also found one calculator for each two participants in workshops to be ideal (McGrann, 1979). The major use of printers is in debugging programs.

A term project of ten to twenty pages plus an oral presentation of the project is required of each student. Students may choose any topic that relates concepts and techniques taught in the course to their personal area of interest. Most students use existing programs or write special programs for the programmable calculator as a portion of their term projects. Each student is given ten minutes in a special interest discussion/laboratory section to present a summary of the term project. Course details can be found in a previous article by the authors (Taylor, 1980).

VI. EVALUATION

To determine student reaction to the course, end of quarter surveys have been conducted for each quarter the course has been taught since adopting the TI-59 as the main educational medium, Spring 1979. The results from Spring and Autumn Quarters 1980 are probably the most representative of the course as now taught. Extensive revisions were made prior to Spring Quarter 1980. All students but two completing the course Spring and Autumn Quarters 1980 wrote evaluations (total of 124 student evaluating).

Students were enthusiastic about using the programmable calculator. All but two rated it as either essential, very important, or important when asked, "How important was access to and use of the TI-59 to your comprehension of computers and their present and future use in agriculture and society?". Eighty-nine percent of all students ranked it in the top five when asked, "Compared to all other courses you have taken at Ohio State, how would you rank "Computers in Agricultural Decisions" as to its use in preparing you for your future?". Students also found the course demanding. When asked, "Compared to all other courses you have taken at Ohio State, how would you rank "Computers in Agricultural Decisions" as to the amount of effort required to meet course objectives?". Eighty-seven percent ranked it in the top five. Students reported spending an average of 66 hours studying outside of the classroom on the five-hour course with very little difference noted for class rank. Sixty-six percent of the students felt the course should be required for all agricultural majors.

Students used extensively the programmable calculators. During Spring and Autumn Quarters 1980, they reported spending an average of 34 hours in the agricultural library operating the units in addition to about ten hours of hands-on use during the discussion/laboratory periods. It is estimated that between five and ten students purchase their own units each quarter.

Teaching fundamental computer technology and subject matter applications by this method has proven to be cost effective. Total cost of equipment for the course, expanded to the ideal, would be about $9400 (Table 1). Annual cost is estimated to be about $3600 (Table 2). Based on average use of 44 hours per student, the direct cost per hour was about 32 cents.[1] This 32 cents per student hour is a small fraction of the cost of providing hands-on experience through time-sharing, typically 5 to 30 dollars per user hour. Indirect costs such as library expenses and electricity were not included.

VII. CONCLUSIONS

Recent developments in programmable calculator technology added a needed dimension to teaching computer technology. Programmable calculators provide educators a means to provide students cheaply with almost unlimited hands-on experience with both computer hardware and software. Programmable calculators provide an excellent tool for teaching basic computer concepts and applications in various subjects. Students can write programs to fit specific applications or use programs already developed. Courses taught using programmable calculators can provide students with knowledge and skills which serve them well in other courses and after graduation. This technology allows subject matter professionals to teach fundamental data processing techniques to their students using data and information with which the students are familiar.

The authors are convinced that teaching fundamental computer technology in a subject matter environment will encourage larger numbers of students to become acquainted with modern data processing.

Table 1. Capital Investment for Data Processing Equipment and Costs for Depreciation, Repairs, and Replacement

Item	Cost[a]	Life Length	Yearly Depreciation[b]	Yearly Repairs and Replacement[c]
		Years		
Programmable calculators (Texas Instruments TI-59) 25 @ $204.00	$5,100.00	5	$1,020.00	$765.00
Printers (Texas Instruments PC100C) 25 @ $148.00	3,700.00	5	740.00	555.00
Agricultural module (Texas Instruments/ Iowa State) 15 @ $37.50	562.50	5	112.50	84.37
Extension Cords	50.00	5	10.00	7.50
Total	$9,412.50		$1,882.50	$1,411.87

[a] Actual 1979 costs.
[b] Zero salvage value.
[c] 15 percent per year.

---

[1] This cost can be cut to about 20 cents per student hour of use if the units placed in the library or learning center are also used in the discussion/laboratory sections.

Table 2. Annual Cost, Cost per Student Hour Taught and Cost per Hour of Hands-on Experience

Item	Annual Cost	Cost per Student Hour Taught[a]	Cost per Hour of Hands-on Experience[b]
Fixed Costs			
Depreciation	$1,882.50	$1.48	$0.17
Repair & Replacement	1,411.87	1.11	0.12
Variable Cost			
Printer paper, 100 rolls @ $3.40	340.00	0.27	0.03
Total	$3,634.37	$2.86	$0.32

[a] 85 Students x 5 hours x 3 quarter = 1,275 hours.

[b] Estimated at 11,220 hours per year or 44 hours per student.

References:

Aronofsky, Julius S., Robert J. Frame and Elbert B. Greynolds, Jr. Programmable Calculators-Business Applications, New York, N.Y.: McGraw-Hill Book Company, 1978.

French, Charles E. "University Uses of Computers-The Purdue Agricultural Case." Proceedings of the First International Conference on Computer Satellites in Agriculture. The Ohio State University, Columbus, Ohio, Oct. 30-Nov. 2, 1972: 139-146.

Fuori, William M. Introduction to the Computer-The Tool of Business. Third Edition, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1981.

McGrann, James M. and William M. Edwards. "Application of the Programmable Calculator to Extension Agricultural Management Programs: Experience from Iowa." North Central Journal of Agricultural Economics 1 (1979):147-153.

Shaffer, Richard A. "The Superchip-Vast Computing Power is Seen as More Circuits Squeeze on a Tiny Part." The Wall Street Journal, LIX, No. 136. Apr. 27, 1979.

Taylor, R.D. and H.W. Newland. "Computers and Economic Animal Nutrition." Proceedings of the First International Symposium on Feed Composition, Animal Nutrient Requirements and Computerization of Diets. Utah State University, Logan, Utah, July 11-16, 1976: 7-14.

Taylor, R.D. and Michael W. Woolverton. "Special Course Report - Teaching Computer Technology for Agriculture Using Programmable Calculators."

Texas Instruments, Incorporated. Programmable TI-58/59-Solid State Software Libraries and Other Accessories. Lubbock, TX: Texas Instruments, Incorporated, 1977.

AN EDUCATIONAL COMPUTER NETWORK FOR THE USERS OF CHEMICAL INSTRUMENTATION

Dr. Michael A Pleva
Dept of Chemistry
Washington & Lee University

Dr. Frank A. Settle, Jr.
Dept of Chemistry
Virginia Military Institute
Lexington, Virginia 24450

AN EDUCATION NETWORK FOR THE USERS OF SCIENTIFIC INSTRUMENTATION

In the current environment of rapidly emerging technology, developments in instrumentation are opening new areas of pure and applied science. Activity in existing areas is also stimulated by these developments. Scientists and engineers are finding it necessary to become well-versed in instrumental methods relating to their work.

A variety of techniques is available to persons wishing to increase their knowledge of instrumental methods. These include the traditional live, stand-up courses, audio-type courses, film/video courses as well as textbooks, scientific journals, and trade publications. Nevertheless, despite all the educational methods employed, advances in instrumentation continue at such a rapid pace that the slower dissemination of information still limits productivity in many areas of science and engineering.

The computer, which is responsible for many recent advances in instrumentation, offers a solution to the problem of rapid dissemination of educational materials to a large audience with diverse interests. It is predicted that by the end of this decade virtually all industries, laboratories, and educational institutions as well as over one-fourth of all U.S. households will be outfitted with microcomputers and auxiliary telecommunications equipment. The resulting information industry should expand into a $500 billion business by the turn of the century.

The Science Education Directorate of the National Science Foundation, recognizing the need for keeping members of the scientific and engineering communities current in instrumental methods, has funded Project SIINC (Scientific Instrumentation Information Network and Curricula). The project, begun in January 1980, is a three-year feasibility study of the use of computer technology to deliver educational materials to people using chemical instrumentation. In this first phase of the project a limited number of modules is being developed. Each module treats a single instrumental method. Initially, these modules will be stored on disks of a Burroughs 6800 computer system at Virginia Military Institute and accessed through dial-in telephone lines. Modular materials may become available on floppy disc for use on popular microcomputer systems. Videodiscs may ultimately offer the best mode of dissemination.

The initial modules available to network users will feature instruments required by the Environmental Protection Agency to analyze priority pollutants in drinking water and in waste water. These methods include gas-liquid chromatography, atomic absorption spectroscopy, gas chromatography/mass spectrometry, high performance liquid chromatography, and inductively coupled plasma emission spectroscopy. Although all these techniques are employed to analyze water for priority pollutants, each instrument module may be used independently by users with other interests.

Materials in the modules are being prepared by task forces of persons knowledgeable in the selected instrumental methods. Three task forces have been organized:

Gas-liquid chromatography; Prof. Harold McNair, Department of Chemistry, Virginia Polytechnic Institute and State University, Blacksburg, VA.

Gas chromatography/mass spectrometry; Prof. Frank Karasek, Department of Chemistry, Waterloo University, Waterloo Ontario, Canada.

Atomic absorption spectroscopy; Mr. Theodore Rains, Research Chemist, the National Bureau of Standards, Washington, D.C.

Each task force includes persons from academia, industry, government and instrument manufacturers.

The system should help practicing professionals in chemistry, other physical sciences, biological sciences, and engineering. They can refresh or update their knowledge of instrumental techniques. The network should help persons assigned to new work areas catch up with current practice. Instructors of undergraduate courses and personnel of smaller laboratories, both of whom are often responsible for a variety of instrument methods, should find the system valuable. Students receiving formal education in either undergraduate or graduate institutions could use the system to supplement information available in textbooks. Managers and other persons responsible for planning should find the system useful for quickly providing the background information required to select and purchase instrumentation.

The program that handles the module files has been designed to make the system simple and easy to use. It has been written using Basic, which offers the following advantages:

(1) facility in programming,
(2) ease of handling strings, even with the primitive form of Basic available on the Burroughs system,
(3) ability to change file assignments within the program, permitting module contents to be divided into a large number of relatively small files. (Both file access by the user and file editing by the task forces are facilitated by this feature), and
(4) transportability of the Basic language.

Most instructions necessary to operate the program appear as prompts to the user while the program is running. A short user guide provides an overview of the system and describes its features. No short courses or seminars are required to train personnel in the use of the system. The designated user concept is thus eliminated; the system is readily accessible to anyone wishing to explore a given instrumental method.

The files forming the nucleus of the system have been prepared assuming that the user has a general introduction to chemical instrumentation at or beyond the junior level of college. File contents provide concise, well-organized discussions and references for many types of users. In this phase of the project, illustrations, diagrams, and data presentations will be limited to character-coded graphics. Future modules recorded on videodiscs may allow the presentation of illustrations and diagrams. Thus the role filled by the system lies between conventional computer-assisted instruction and information retrival from a large data base, such as the Lockheed Dialog Service.

All of the files in any module are stored under one of the following general topics:

(1) How It Works (Basic phenomenon measured, instrument components, instrumental operating principles, simple teaching experiments)
(2) What It Does (Qualitative and quantitative analysis, extent of automation possible)
(3) Nuts 'n Bolts (Major suppliers, maintenance and operational requirement, accessories, required training, annotated bibliography)
(4) Illustrative Case Studies Of Applications
(5) Newest Innovations

The exact number of files and the presentation of material within the files are determined by the individual task forces.

Module files may be accessed using several pathways (figure 1): (1) the tree mode, (2) index mode, or (3) the keyword mode. The tree mode emphasizes the educational aspects, allowing the user to make choices that direct him to the desired information. Persons with minimal background in the selected instrumental method will most likely access the files using this mode. The most direct method of accessing the files is the index mode. In this mode, the user is presented with the table of contents for the module selected. The table of contents contains a title and corresponding code for each file. The chosen file is accessed by entering the appropriate file code.

In the keyword search mode, all files of a given module are searched for a keyword or key concept. A list of these keywords and concepts with their corresponding two-letter codes is displayed upon initiation of the search mode. When the user enters a two-letter code, the search starts. Upon completing the search, the name and corresponding five-character code for each file containing the keyword are listed. The user then enters the desired file code, whereupon the file is searched until the line containing the keyword is located and displayed. File manipulation options allow the user to examine the context of this part of the file in more detail.

In all modes, once the file code has been entered and a file is being read, the

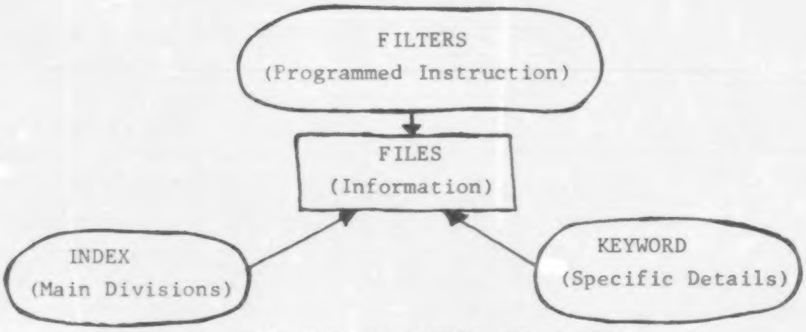

FILTERS
(Programmed Instruction)
FILES
(Information)
INDEX
(Main Divisions)
KEYWORD
(Specific Details)

Figure 1 WAYS TO ACCESS THE SYSTEM

following options are available at the end of each page (20 lines) of file text:

(1) display the next page.
(2) jump forward by a specified number of lines.
(3) jump backward by a specified number of lines.
(4) access the module glossary and return to the location in current file from which the glossary was called.
(5) exit before the text is exhausted.

When the user has finished reading the files, the program permits recycling to a new mode. If none of the available options is selected, the program requests the user's name and address. The user may then enter comments via the message board, a useful feature in evaluating current files and in creating new files addressing topics requested by users.

The user can also request printed copies of each file. At the end of the session, the selected file contents are sent to a line printer at the VMI Computer Center and then mailed to the user.

To evaluate the use of the system and provide requested information, system use is monitored to give the user's name, elapsed time of session, modules accessed, mode used, and what files were read. This information, along with the user's comments, has been valuable in determining which files should be modified and which new files should be added. Task forces are planning to revise current files and add new ones once a year.

At present, the gas-liquid chromatography module of the system is being evaluated by approximately forty project associates. File materials are being prepared by the atomic absorption spectroscopy and gas chromatography/mass spectrometry task forces. The first versions of these modules should be available to users by the fall of this year. A Burroughs 6800 computer with 2 million bytes of core memory and a single 87-megabyte disk appears tobe adequate for the operation of this phase of the project.

In summary, electronic storage and dissemination of instrumental educational materials offer major advantages over methods currently employed. Materials in the files may be updated easily, allowing the system to remain current. Input from users and monitoring of system operation provides the bases for addition, deletion, or modifications of files. Multipath access to module topics, provision for rapid scanning of files, and simple graphics are features that offer the speed, flexibility, and responsiveness necessary for the wide-spread use of the system.

References

M.A. Pleva and F.A. Settle, American Laboratory, 5 (1980) 95.

F.A. Settle and M.A. Pleva, Journal of Chemical Education, 57 (1980), A255.

NEWTON - A MECHANICAL WORLD

Alfred Bork
Stephen Franklin
Martin Katz
John McNelly
Educational Technology Center
University of California
Irvine, California 92717
(714) 833-6911

This paper reports on an aid for learning classical mechanics developed at the Educational Technology Center at the University of California, Irvine. The simulation, the type we call a controllable world, has gone through a number of stages and variations. Its primary role is to improve intuition of students in the beginning quarter or semester of a physics class. Although our use of programs such as this has been at the beginning college level, the program is suitable for high school use.

Controllable Worlds

A controllable world is a computer program providing a world in which the student can move around freely, exploring at will. The idea is related to play activity as seen with young children. A rich collection of facilities is provided, and the learner has freedom in using these. A key to the success of such a controllable world is to make its use as easy and as obvious as possible.

We do not provide structured learning in such an activity. Rather, we are concerned with the experiential phase which might precede a more formal learning approach, as described in the learning cycle of Robert Karplus.[1] The ease of use cannot be overstressed. If the student is to be encouraged to experiment, to try many things, it must be easy to do so. Thus, controllable worlds should incorporate a wide variety of vocabulary rather than restrict the learner to a very specialized menu of choices. They should forgive typing errors as far as possible. They should let the student know what is happening at each stage. They should be visual, because the visual world is for many students a more interesting, exciting and intuitive world than the world of numbers and words.

The main role of the controllable worlds as developed in the Educational Technology Center over the past dozen years is to increase student intuition and insight into the phenomena. Courses often put too much stress on formal manipulation, the skills that are needed to solve particular classes of problems. But often students do not develop an intuitive background that leads to imaginative solutions of new types of problems. The controllable world can provide a rich collection of experience which can lead to such intuition. But it does not necessarily do so,[2,3] as we will comment later.

NEWTON

The controllable world discussed is based on Newton's laws of motion. The computer "knows" that $\vec{F} = ma$, and furthermore knows the mathematical tools necessary to turn this into visual information about how bodies move. The student is given plotting capability. After the force is chosen, the learner can plot various physical variables against each other, alter initial conditions and constants in equations, and move freely through the program. Newton is self-explanatory, not dependent on print material. But certain types of print material will typically be used by students with the program.

History

A program of this type was described in the initial proposal from the University of California, Irvine (to the National Science Foundation) for developing graphic computer-based learning material. Shortly after the grant, Richard Ballard joined Alfred Bork on the project. They developed the initial version of a controllable world called Motion. This program, in a time-sharing environment, is still used with beginning physics students at Irvine. Motion went through a number of variations, as we experimented with how it could be used most effectively.

Several years ago the focus of development at Irvine began to move from time-sharing to the newer personal computer environment. At that time we also abandoned earlier software approaches, as they were no longer in keeping with what was known about the art of complex programming. Our new developmental language is Pascal, under the UCSD Pascal system.

Martin Katz, then an undergraduate student working with the Educational Technology Center, developed a Pascal version of Motion soon after we began to use Pascal. This version was not completely equivalent to Motion; it omitted some facilities but had some additional ones. This program eventually evolved into Newton.

The current version of the program was developed by Alfred Bork, Stephen Franklin, and John McNelly. It does not follow all the details of Motion. Rather, we tried to use what we had learned in the many years of using Motion with sizable numbers of students to guide the development of the new program. Motion ran on Tektronix displays. Newton was developed on the Terak 8510/a. By and large, we found that the advantages of the personal computer far outweighed the disadvantages; that is, the switch from time-sharing to the personal computer was primarily a gain. We gained selective erase capabilities and better control over timing issues at the expense of poorer resolution.

As of this writing, the latest version of Newton runs only on the Terak. However, earlier versions were successfully moved to the Apple, and we expect to eventually run on a variety of other personal computers. As with other recent developments at the Educational Technology Center, we find it convenient to develop materials on a more powerful machine than the eventual delivery machines.

Capabilities of Newton

As already suggested, Newton is primarily a plotting program. After the mechanical system has been picked, the student can ask to plot any two or three mechanical variables. Time in each case is the independent variable, as is generally the case with mechanical systems, but time does not need to be one of the variables plotted. The user can change the force, change the contants in the force law, change the initial conditions, choose what to plot (including functions of the variables), and query the system for various information. Control over scaling is also available. These capabilities will now be described in more detail.

1) Choosing the Force: When the program is initially entered, the learner must choose a force. At any time during the program, a NEW FORCE can be requested, and the choice will be offered again.

Two basic options in choosing a force are available. First, built-in forces can be picked. Currently the built-in forces are gravitational motion with one force center, gravitational motion with two force centers, simple harmonic motion, and forced driven harmonic motion. New built-in forces are being added. Built-in forces can be selected from a menu.

The user can also choose to enter almost any force whatsoever. These are accepted in a typical linear computer algebra form with some flexibility. The computer queries for each component of the force. In specifying the force, if constants are used that are not known to the system, the system will query the user as to what initial value should be assigned to these constants. The program can handle almost any force within the limitations of typing.

2) Plotting Capability: After a force has been chosen, either initially or at some later time in the program, the machine is prepared to plot something. That is, if the user simply types PLOT a curve will appear. The curve is dependent on the force law chosen. We have chosen in advance an interesting case with all the initial conditions already chosen.

Many computer simulations query students for everything necessary to plot. Beginners seldom understand what things are necessary or what values to assign to them, so such querying should be delayed until the learner has attained better understanding. Our notion is to provide an interesting case to begin with and allow the student complete control over changing each of the variables involved.

If the student wants to plot two different variables, then the command is

```
PLOT X,T      or
```

GRAPH X VS Y

Other forms are also possible. Newton is flexible about what terminology is needed, often providing alternates. One function of the variables can be plotted against another.

The typical way to stop plotting is by pressing the space bar. One can continue plotting by typing CONTINUE PLOT after such a stop. There are certain circumstances where plotting may stop on its own, such as when the body crashes into a sun, or the values calculated become abnormally large.

A number of other capabilities are associated with plotting. One that is frequently useful is OVERPLOT, which allows learners to keep a previous curve while seeing a new one. If parameters change, it is convenient to see the curves before and after the change.

The current values of variables plotted can be determined at any time by typing a question mark. This is often convenient when you need numerical values in addition to the curve.

3) Querying for Information: The user can ask for information about current values of variables, either before or after a plot. Thus, you can ask for INITIAL CONDITIONS and Newton will give you these values. Or if you want to see all the variables associated with a particular case, you can simply type ALL to see them.

Such information is often more detailed than necessary. Usually learners need to know only certain variables, either the initial conditions or some of the constants in the equations representing the force law. The student can ask directly for these in one of the following fashions:

a) WHAT IS X?

b) X = ?

c) X ?

Newton will regard any of these commands as equivalent.

4) Changing Variables: A user can not only query variables; variables can be altered. This is done by entering small assignment statements:

a) X = 3

b) FRED = 2, or

c) DT = DT/2.

In the second case it is assumed that a user specified force was used and that the variable FRED was picked by whoever entered the force. When a variable is changed, Newton verifies what the user has done by showing the value of the variable as changed. This is often not necessary, but it is a reasonable precaution to overcome typing errors.

5) Changing the Scale: Often it is necessary to modify the scale of the plotting to see a convenient picture on the screen. This must be done by the user, as Newton cannot know what details the user expects to see.

The overall changes of scaling are indicated by the commands MOVE BACK and MOVE CLOSER. Both of these produce a scale factor change of 2 on both axes.

Direct scaling is possible by plotting variables in the following way:

PLOT .5 * X*X, 3 * VX

The net effect will be that the scales of both variables change independently. Axes will be labeled appropriately, reminding the user of this change.

Uses of Newton

We have implied that the development of materials and their effective use in class or learning environments are two quite separate issues. In this section we discuss this situation and clarify the use of Newton and similar programs.

The primary use of such a program is to build intuition, to allow learners to gain a range of experiences that are not present in everyday life, and so have a feel for mechanical systems that goes beyond the ability to manipulate mathematical details to obtain solutions. Simulations, such as the present one, often have an immediate appeal to scientists. They are closest to the directions scientists follow in their professional activities using the computer. Most scientists are stimulated by running such simulations. Indeed, in our early days with Motion, scientists could hardly keep themselves away once they became exposed. We would have visitors spend large amounts of time running it. Motion would also draw very large crowds when presented at professional meetings.

We began to understand the distinction between the program itself and the program operating in a classroom when we began running Motion with sizable groups of students. Here the excitement of the scientists was often not present. Only a small percent of the students would become excited while using the program. The rest would use it a brief time and then stop unless forced to continue. While we thought the program exciting, the bulk of students in any large class did not appear excited.

This situation puzzled us until we began thinking about incorporating the material in the classroom. Our first step was to develop computer exercises that assured each student would see at least the most important experiences. These computer exercises are still in use in the time-sharing environment, about six years after their initial development. Here is a sample of one exercise concerned with gravitational motion:

> Now you are to see what would happen if gravitational force were *not quite* inverse square. Ask for the EQUATION again; the power is N. Set
>
> N = -1.9
>
> Return to plotting the X-Y- space, investigating a range of values around -2. You may want to continue plotting each orbit. What can you say about the results? What happens for values less than 2? Greater than 2?

> What happens if we examine behavior in velocity space?

> Now consider the case of *two* gravitational force centers, as if you had two *fixed* suns. Request
>
> 2 FORCE CENTERS
>
> at any input. The initial conditions will be reset. Determine them by typing
>
> X,Y,VX,VY = ?
>
> PLOT the orbit. Discuss the possibility of life on a planet with such an orbit.

> See if you can find velocities that give closed (repeated) orbits. What velocities do this? Sketch the orbits.

We want to make certain that the student has some structured experiences that aids *learning* about how mechanical systems work.

This second round, using Motion with computer exercises, was not entirely successful either. In the evaluation of the course made by Michael Scriven and his colleagues, this was one of the most criticized activities. At this time we made it a required part of the course. We found that students did not see its connection to other material within the course. Now the material is explained better and used as an option with much greater success; it is used by a sizable number of students.

But we do not regard this approach as entirely satisfactory either, because we believe the experiences should be for *all* students. In another controllable world dealing with field lines, we have greater success with a different tactic. We built an on-line quiz around the simulation. The quiz notes if students have developed the insights we expect about the way field lines behave. As yet, we have not followed this same tactic with mechanics but probably will do so.

The computer experiences for Motion only cover some of the areas of beginning mechanics. We could increase the viability of this program by making it a constant component of the beginning course, making every unit depend on it to some extent. We are working with CONDUIT (specifically Arthur Luehrmann, Herbert Peckham, Harold Peters, and Alfred Bork) to develop a more extensive set of computer exercises for Newton. These are intended for use in high school and beginning college physics courses and will cover areas not done so by the present exercises. For example, we consider motion with no forces acting and motion with constant forces. We plan to have these new exercises available at the conference.

This project is supported by the National Science Foundation through a CAUSE grant. The project manager for the grant is Stephen Franklin. Other members of the Educational Technology Center, Barry Kurtz and David Trowbridge, have offered helpful suggestions for developing Newton and the associated exercises.

Newton will be available for demonstration at the National Educational Computing Conference. The talk will be illustrated with many slides showing its function in more detail. The Educational Technology Center is happy to have visitors and is quite willing to send additional information about its activities.

References

1. Karplus, Robert, "The Psychology of Teaching and Thinking for Creativity," Anton Lawson, ed., 1980.

2. Bork, A., "Computers as an Aid to Increasing Physical Intuition," Am. J. Phys. 46 (8), August, 1978.

3. Bork, A., "Learning with Computer Simulations," Computer, October 1979.

4. Bork, A., "The Physics Computer Development Project," EDUCOM, Vol. 10, No. 4, Winter 1975.

DESIGN OF COMPUTER-BASED INSTRUCTIONAL MATERIALS TO ENHANCE THE PROBLEM-SOLVING ABILITIES OF SCIENCE STUDENTS

Jerry P. Suits and J. J. Lagowski
Chemistry Department, The University of Texas at Austin
Austin, TX 78712

The purpose of this paper is to describe how computer-based instructional materials, CBIM, may de designed to enhance the problem-solving ability of science students. Science students need this ability in order to participate in meaningful problem solving which leads to understanding the concepts and principles in a particular scientific discipline. Memorizing the algorithm of a computational problem or defining a scientific concept does not substitute for ability to solve a novel problem, Richardson (1). These materials must contain all the characteristics of high-quality CBIM programs (see first section). Also, the characteristics of meaningful problem solving must be included in the materials (see second section). In the third section, an example of a CBIM problem-solving lesson actually used in a science course at The University of Texas at Austin will be described in terms of these CBIM and problem-solving characteristics.

CHARACTERISTICS OF CBIM PROGRAMS

The essential characteristics of CBIM programs which facilitate student learning have been well documented; Dence (2), Culp (3), and Wade (4). Wade lists five fundamental characteristics: [1] the learning must be congruent, [2] the learner must be ready, [3] learning needs to be managed, [4] the learning tasks must be attainable, and [5] learning must be efficient (4).

[1] The learning must be congruent (5) with the philosophy of the instructional system overall. The coordinated efforts of an expert with the instructional designer can ensure this harmony. In addition, the instructional unit must contribute to the general goals of the course; it must be accurate, up-to-date, and sufficiently complete to achieve its purpose. EVALUATION by the students and appropriate tests may lead to revisions which establish and maintain these requirements.

[2] The learner must be ready in the sense that he must possess the necessary intellectual skills, the ability to manipulate the information. If the learner lacks the necessary skills, then branching to a TUTORIAL SEGMENT may be used to develop them. However, if he has not developed formal reasoning ability, then other factors must be considered, Inhelder (6). A learner will be ready emotionally if he possesses a high degree of curiosity with a correspondingly low level of anxiety. REINFORCEMENT supplied by the CBIM program at appropriate times will foster this emotional readiness. Finally, physical readiness may decline if the learner is forced to sit at the computer terminal for over 45 minutes (3).

[3] Learning needs to be managed during the entire sequence of instruction from the presentation of INSTRUCTIONAL OBJECTIVES to the FEEDBACK message given upon completing the LEARNING TASK specified by the objective; Dence (2), Culp (3), and Wade (4). The INSTRUCTIONAL OBJECTIVES establish a learning set in the student which predisposes him to selectively attend to certain classes of stimuli while ignoring others. The program randomly or selectively GENERATES a question, problem, tutorial material, or graphics to which the student responds. The student input is READ and INTERPRETED followed by DIAGNOSTIC BRANCHING based on student response, reinforcement if correct, corrective hint if recognized incorrect, detailed answer if several tries have been made, or skip to next question if requested. During the entire sequence CUES should direct attention to the relevant aspects of display stimuli. IMMEDIATE FEEDBACK is given in the form of the correct answer, a correct or incorrect message, or response contingent tutorial information.

[4] Learning tasks must be attainable for the learner to assimilate the information presented into his existing cognitive

domain. Information should be displayed logically only when the student is ready to process it. Short-term memory should not be overloaded because it can hold only one piece of information at a time and because coding for permanent storage requires a certain period of time. Information which is to be stored in long-term memory requires sufficient time and the related knowledge to be initially encoded and then decoded upon retrieval. Programs should judiciously use timed messages when students are slow on higher level cognitive activities.

[5] Learning must be efficient since the student's time should be used conservatively. If the instruction takes the characteristics of the individual learners into account, then learning efficiency is maximized. Programs which provide for diagnostic branching would obviously maximize efficiency.

CHARACTERISTICS OF PROBLEM SOLVING

While the characteristics described in the above section are sufficient for nearly all instructional computing needs, additional characteristics must be defined for programs designed to enhance meaningful problem solving (7). The learner must be in control of certain features of the meaningful problem solving situation, whereas the designer must provide guidance on features which direct the learner to a solution. Learner control of all instructional tactics has been effective on lessons requiring only algorithmic problem-solving abilities in highly structured job areas,Steinberg(8). Conversely, program control has been found to be more effective on more complex learning tasks, Tennyson(9). Meaningful problem solving is a more complex task, thus complete learner control seems inappropriate. Some learner control on complex tasks is necessary because meaningful problem solving involves the learner's skillful blend of procedural and declarative knowledge. Therefore, instruction that emphasizes both procedural and declarative instruction should help students transcend the oversightedness of the learning sets activated by each kind of instruction, Richardson (1).

Program Control

All five fundamental characteristics of CBIM programs are pedagogical decisions made by the program designer. To aid meaningful problem solving, the designer should [1] determine the pedagogical sequence, [2] sequence the objectives to establish a learning set, [3] provide numerical cues for the primary instructional sequence, and [4] embed science process skills in the structure of the problem-solving task.

[1] The theoretical basis determining the PEDAGOGICAL SEQUENCE of the problem-solving task is two fold:

[a] First, according to Bruner the mode of representation should begin with enactive representation (learner actively does something) followed by iconic representation (learner observes a set of summary images or graphics), then concludes with symbolic representation (a hierarchical set of symbolic propositions or concepts) (10). An example of the principle of an equal-arm balance illustrates this sequence of representations. Young children maneuvering themselves on a seesaw is an enactive representation of the principle. Older children arranging rings on a two-arm balance is an iconic representation. A person using Newton's law of moments or a verbal description of the principle is a symbolic representation. Bruner states that the domain of knowledge is understood with the least quantity of data when learners follow the above sequence. He also states that this instructional sequence should "increase the learner's ability to grasp, transform, and transfer what he is learning" (10).

[b] The second basis for PEDAGOGICAL SEQUENCING is based on the placement of the rule and practice within the structure of the problem-solving task. When the rule (concept or principle) is placed before the practice, the instruction proceeds from general to specific, the deductive method. Conversely, when the rule is placed after practice, instruction proceeds from specific to general, the inductive method. Which method is better is dependent upon the theoretical assumptions made by the designer. Bruner and Piaget state that meaningful learning is facilitated when experiences with the senses precede the mathematical or verbal statement of the rule (11). On the other hand, Ausubel believes that statement of the rule internalizes the rule in its final form in the learner's cognitive structure which is more efficient than having the learner discover the rule from specific cases (12). Evidence from research studies does not consistently favor one method or the other; Tanner (13), Hermann (14), and Merrill (15). We are currently analyzing data to compare these two methods used in CBIM programs designed to provide first semester chemistry students with simulated experiments which develop chemical concepts or principles, Cavin (16).

[2] The sequence of procedural and declarative PROBLEM SOLVING OBJECTIVES may have an effect on the learning set and subsequent performance routine of the student.

If the student perceives that he will do several procedures and then derive meaning from the combination of procedural results, he should invent a performance routine which is matched to the teaching routine. This coupling of routines should help transform the student's experience of instruction into an ability to use his experience (1). In other words, he has acquired an ability to engage in meaningful problem solving. Instructional design should [a] pose the problem, [b] list procedural objectives, and [c] conclude with a declarative objective which ties the set of procedures together. Sequencing procedures which require action before the declarative verbal or mathematical equations are consistent with Piaget's and Bruner's theoretical constructs.

[3] The basis for the PRIMARY INSTRUCTIONAL SEQUENCE are the steps in the laboratory experiment in which the learner goes through a complete series of steps (in proper order): [a] statement of problem and objectives; [b] prelab activity and quiz provide the necessary cognitive skills; [c] the simulated experiment generates data which are internally consistent within experimental error; [d] in the data analysis, the data are reduced by a mathematical relation or logical statement which is a compendious resultant; [e] the data significance or explanation provides a pattern or rule which can then be used to predict future experimental values.

Learner Control

Learner control consists of four levels of conscious processing over which the learner can control task-related decision making: [1] display selection, [2] conscious cognition, [3] content selection, and [4] learning set, Merrill (17).

[1] DISPLAY SELECTION means the learner decides what type of presentation he wishes to study next. It is an option in any individually studied material. CBIM programs should not embed individual components in the surrounding text so that it requires considerable searching for the student to isolate, select, and sequence the individual displays (17). Use cues to direct attention to important aspects of a display, such as key words, or critical structural features, Koran (18). There are two types of display sequences. A cummulative display sequence begins with a fundamental display and builds upon it cummulatively. A noncummulative display sequence consists of a set of displays in an instructional segment which are relatively independent of each other. Learner control should be maximized in the latter case but minimized in the former.

[2] CONSCIOUS COGNITION refers to those deliberate mental activities that a student employs to remember and integrate new information. Students must compensate for a wide variety of displays by manipulating their own conscious cognitive processes to learn effectively. A list of these processes includes: rehearsal, repeating a message from memory; repetition, expressing information in learner's own words; imaging, forming mneumonic from information; covert practice, mental questioning and answering; and analogy, parallel of the information from some other subject area (17). Conscious cognition is normally independent of external control, but Merrill and Callahan have evidence that certain task strategies aid learning (17).

[3] CONTENT SELECTION is the decision by the learner about what instructional segment to study next. Content selection is maximized when the learner can select from a large array of segments and minimized when the program or instructor sequences the segments into lectures or linear programs (17). Deciding which element, the program or the learner, should sequence the content depends on the nature of the prerequisites in the learning task. A lesson with intrinsic prerequisuite segments is nonarbitrarily ordered by the attributes of the subject matter. For example, in a science laboratory exercise a person must gather the experimental data before he analyzes it. To reverse the two segments is meaningless. A lesson with extrinsic prerequisite segments is dependent upon the knowledge possessed by the learner prior to the lesson. For example, if a student in physics knows the formula for Newton's second law of motion, $F = m * a$, and is given the values of mass and acceleration, then he does not need to see the rule display which shows the formula. However, the learner who does not know the formula must have the rule display before he can solve the problem.

[4] LEARNING SET is an activated cognitive structure that distinguishes relevant features and processes in a lesson, Mayer (19). A learning set thus guides the learner through the instruction so as to avoid overstimulation from the instructional environment and misdirections due to potentially divergent features or processes. For example, a person may be overstimulated by the large number of pieces or color and shape when trying to piece together a jigsaw puzzle without a photograph of the whole picture. Misdirection due to divergent features might occur when the puzzle solver tries to piece together a portion of the reflecting-pool image of

the snow-covered mountain with pieces of the actual mountain.

Learner control over all four levels has been effective for algorithmic problem solving, Steinberg (8), but not for meaningful learning, Tennyson (9). Thus, in meaningful problem solving the learner and program control is integrated. We propose that the localized features of learner control, i.e. display selection and conscious cognition, should be at the discretion of the learner, whereas the directional features, i.e. content selection and learning set, should be controlled by the program. With this integration the learner does not have to generate an extraneous learning set to discriminate between the directive hierarchy of cognitive structures and the localized hierarchy of cognitive structures. A novice learner (the only real kind of learner) should not be expected to organize two different kinds of problems, i.e. localized structures for the subject matter problem and directive structures for the instructional problem. If the same format and sequence are used repeatedly to solve different content problems over a period of time, then the student can internalize those cognitive structures which could allow him to do open-inquiry problem solving. Meanwhile, CBIM lessons which address instructional objectives with a meaningful problem-solving task should integrate learner and program control in order to enhance the problem-solving abilities of science students.

AN EXAMPLE OF CBIM PROBLEM SOLVING

In this section an example of a computer-based instructional problem solving, CBIPS, lesson which was actually used in General Chemistry I, a lecture course at The University of Texas at Austin, is described. The features, sequences, and instructional strategies which reflect the characteristics delineated in the two previous sections on CBIM and meaningful problem solving are emphasized.

The particular CBIPS lesson selected was the first in a series of ten computer lessons used over the entire Fall 1980 semester by a large section of General Chemistry I students. The overall goal of "Computer Lesson # 1: Pure Substances and Mixtures" was to use the shape of a heating or cooling curve to classify an unknown sample of matter as a pure substance or mixture. The instructional objectives of the lesson reflect the type of learning and sequence of instruction:

Objective 1: Upon selecting the complete set of simulated experimental conditions, collect time, temperature, and physical state data over a range of twenty degrees above and below the melting point of your sample.

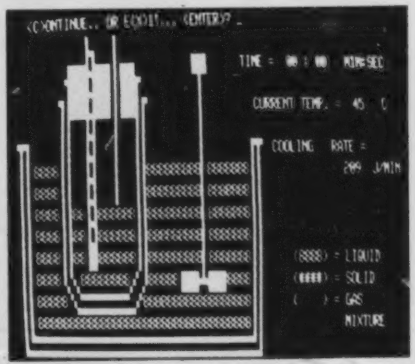
Objective 2: From a computer-drawn plot of your time and temperature data, determine the melting point of your sample. (Melting point is the temperature at which the last portion of solid has just liquified.)

Objective 3: Given the general shape of the heating or cooling curves for a pure substance and a mixture, classify your sample as a pure substance or mixture.

Objective 4: State the basis for your classification above.

This lesson demonstrates a generality called higher-order principle: pure substances melt or freeze at a constant temperature, whereas mixtures melt or freeze over a range of temperatures.

Notice that the objective sequences consist of three procedural objectives followed by a declarative objective in which the student attaches meaning to the set of procedures. This sequence should activate a learning set which is parallel with the overall direction and goal of the lesson (see subsection on learning set). The representation of the generality begins with an enactive representation, selecting experimental conditions, then observing the time, temperature, and physical state (see Figure 1). The student observes an iconic representation: a plot of time vs temperature characteristic of the type of unknown (see Figure 2) and then points to the curve shape characteristic of his unknown (see Figure 3). Finally comes symbolic representation, a verbal statement of the generality elicited from the student by the program.



```
<C>ONTINUE.. OR E<X>IT... <ENTER>? _
TIME = 00 : 00 MIN:SEC
CURRENT TEMP. = 45 C
COOLING RATE =
209 J/MIN
(8888) = LIQUID
(####) = SOLID
(    ) = GAS
MIXTURE
```

Figure 1: A liquid sample (inside double test tube) is being cooled by the surrounding cooling environment which is stirred by a motorized paddle wheel; the level of mercury in the thermometer drops with the cooling of the sample.

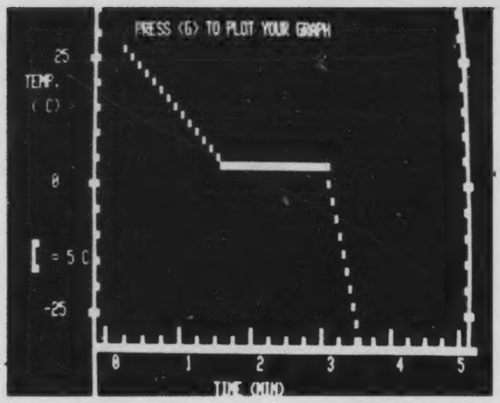

PRESS (G) TO PLOT YOUR GRAPH

Figure 2: Student enters data, then observes the time vs temperature plot after pressing [G].

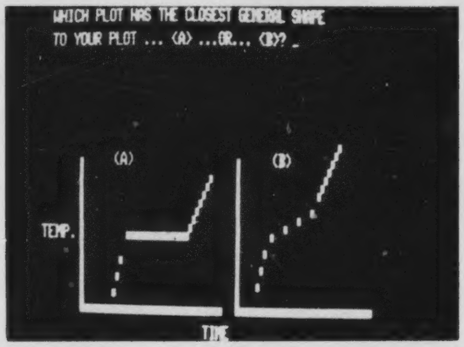

WHICH PLOT HAS THE CLOSEST GENERAL SHAPE TO YOUR PLOT ... (A) ...OR... (B)? _

Figure 3: Student selects the plot which most resembles the shape of his plot.

A verbal statement of this generality may be placed at the beginning of instruction to generate a deductive method. Conversely, the generality may be placed after the elicited student explanation of the relationship of shape of the curve to the type of matter in the unknown to form an inductive method.

This example also illustrates how the characteristics of quality CBIM lessons as described in a previous section are complementary to the characteristics of meaningful problem solving described above. First, the lesson is congruent with the philosophy of chemistry which is an empirical discipline requiring exploration of the interface of experimental results and chemical concepts and principles. Second, the learner is exposed to factors which should induce all three types of readiness: [1] cognitive readiness is required since the learner must pass a prelab quiz in which he does calculations and identifies various parts of the melting-point apparatus (see Figure 1); [2] emotional readiness may be induced by a lower anxiety than in the real experiment because there is no fear of breaking glassware or damaging expensive equipment plus curosity if the learner thinks, "Is my gold ring made of pure gold or gold and something else?"; [3] physical readiness may decline if the student sits at the terminal continuously for the entire 45-75 minute lesson. A better plan is to separate prelab (15 min.) simulated experiment (30 min.), and postlab (15 min.). Diagnostic branching occurs since a student who decides to heat a liquid to find the melting point is given an appropriate message. Also, a student who chooses the wrong type of matter for his curve shape is given the verbal statement of the generality, whereas a student who is correct tries his own verbal statement first. A student is not given a limited response time followed by a display of the correct response since this is a higher-cognitive activity which requires thought time. Some students do the melting point experiment three times just to get a feel for it, whereas others do the experiment, collect the data, and continue. This example of learner control and the above examples of diagnostic branching both represent efficient use of learner time.

The primary instructional sequence of this lesson parallels the steps of the real empirical problem solving experiment. In this lesson the steps are: [1] the four objectives are displayed after the problem is detected; [2] the prelab activity described above; [3] the experimental time, temperature, and physical state data are gathered during the computer-simulated experiment; [4] the plot of time vs temperature in conjunction with changes in physical state allowed the melting point to be determined; [5] statement of the generality provided a rule by which additional samples of matter could be classified as pure substance or mixture (allows prediction). Obviously, many science process skills are embedded in this lesson, such as comparing an observed melting curve with standard melting curves. Although the student was not engaged in hands-on experience with the phenomena, he was engaged in the minds on experience which consists of the equivalent mental processes of the real laboratory experiment, Pavelich (20). He was, after all, doing something rather than being told. This is the best way for learners of any age to learn new information according to Piaget (6).

REFERENCES

(1) Richardson, J. J., 1980 Proceedings: Association for the Development of Computer-Based Instructional Systems, p. 24.

(2) Dence, M., Educ. Tech. 20, 50 (1980).

(3) Culp, G. H., Design and Development of Computer-Based Instructional Materials, (Unpublished manuscript, The University of Texas at Austin).

(4) Wade, T. E., Jr., Educ. Tech. 20, 32 (1980).

(5) Note: The underlined portion of each subsection represents the learner's needs for quality instruction, whereas capitalized words represent conventional CBIM and Systems Approach characteristics.

(6) Inhelder, B., and Piaget, J., The Growth of Logical Thinking from Childhood to Adolescence, Basic Books, 1974.

(7) Note: Richardson describes two types of problem solving: algorithmic problem solving (requires only procedural knowledge) and meaningful problem solving (requires a blend of procedural and declarative knowledge).

(8) Steinberg, E. R., J. Computer-Based Instruction 5, 84 (1977).

(9) Tennyson, R. D., J. Educ. Psych. 72, 505 (1980).

(10) Bruner, J. S., Toward a Theory of Instruction, Harvard Univ. Press, 1966.

(11) Note: Bruner's position is that language plays a central role in cognitive development, whereas Piaget favors hands on experience.

(12) Ausubel, D. P., The Psychology of Meaningful Learning, Grune & Stratton, 1963.

(13) Tanner, R. T., J. Res. Sci. Teaching 6, 136 (1969).

(14) Hermann, G. D., and Hincksman, N. G., J. Res. Sci. Teaching 15, 37 (1978).

(15) Merrill, M. D., Olsen, J. B., and Coldeway, N. A., Courseware Research Report no. 3, 1976, Courseware, Inc., Orem, Utah.

(16) Cavin, C. S., Cavin, E. D., and Lagowski, J.J., J. Chem. Ed. 55, 602 (1978).

(17) Merrill, M. D., Comput. & Educ. 4, 77 (1980).

(18) Koran, J J., Jr., Koran, M. L., and Baker, S. D., J. Res. Sci. Teaching 17, 167 (1980).

(19) Mayer, R. E., J. Educ. Psych. 67, 331 (1975).

(20) Pavelich, M. J., and Abraham, M. R., J. Chem. Educ. 56, 100 (1979).

MICROSIFT PROJECT AND COURSEWARE EVALUATION

Robbie Plummer
Education Service Center-Region 10
Richardson, TX 75080

ABSTRACT:

MicroSIFT (Microcomputer Software and Information for Teachers) is a central project of NWREL's Computer Technology Program that has been productively involved in developing methods and materials for computers in education for over a decade. The MicroSIFT project focuses on instructional applications for K-12, with a major emphasis on establishing effective procedures for the collection, evaluation, and dissemination of materials and information.

The key objectives of the MicroSIFT clearinghouse are:

1. To develop and implement a model for disseminating microcomputer software, information, and materials for educational use at K-12 levels.

2. To develop, validate, and implement an evaluation model suitable for computer-based instructional packages, which will support long-term activity of assessing the quality of materials proposed for distribution.

4. To develop and implement a feed-forward model to guide and direct development of new computer-based instructional materials.

The MicroSIFT clearinghouse procedures and services have been designed by Judith Edwards, Director of the Computer Technology Program, and Donald Holznagel, Coordinator of MicroSIFT, with significant input from the staff of CONDUIT, the Iowa-based clearinghouse for software for higher education, two MicroSIFT advisory boards (a National Technical Advisory Board and a Regional Users Advisory Board), and other consultants.

THE POSSIBLE EFFECT OF COLOR ON COMPUTER-ASSISTED LEARNING

Chaired by Herb Nickles
Coordinator of Instructional Computing
California State College
5500 State College Parkway
San Bernardino, CA 92407
(714) 887-7293

ABSTRACT:

The use of color in computer-assisted learning (CAL) is becoming more common-place as manufacturers of microcomputers provide inexpensive color output. This session will explore the possible positive or negative effects that color has on a learner's physiological and psychological functions.

Representatives from major micro-computer manufacturers will provide an overview of the technology as it relates to color. They will also discuss their rationale for the inclusion or exclusion of color in a CAL setting.

The physiological effects of color on the human body will be reviewed in relation to electromagnetic spectrum theory. Research in this field indicates that some colors can raise the pulse rate, increase respiration, blood pressure and perspiration, while other colors can lower these body functions and increase concentration. Other research has shown that both scholastic and work performance can be effected by colors in the surrounding environment. The implications of this research on CAL will be discussed.

Color also provides a useful dimension for the display of information in a CAL situation. It provides a degree of freedom, or if used properly, degrees of freedom, to enhance similarities and differences. In addition, it inspires motivation. Both good and bad techniques for using color will be explored.

PARTICIPANTS:

Carin Horn
Department of Computer Science
North Texas State University
Denton, TX 76203

Kristina Hooper
Department of Psychology
University of California
Santa Cruz, CA 95060

Al Ciaglia
Commodore International
4350 Bettwood Parkway
Dallas, TX 752___

William Kernahan
Texas Instruments
P.O. Box 225474,MS372
Dallas, TX 75265

Jimmy Thompson
Tandy Corporation
1 Tandy Center
Forth Worth, TX 76102

PASCAL TUTORIAL

H. P. Haiduk
Instructional Computing Liaison
Amarillo College
P. O. Box 447
Amarillo, TX 79178
(806) 376-5111

ABSTRACT:

This tutorial is designed to provide an insight into the elegance, power, and philosophy of the language Pascal. To accomplish this, the tutorial will consist of the following:

1. brief historical view of Pascal in terms of its philosophy and stated design goals;
2. brief review of its current relevance in a diverse set of applications, particularly as it may relate to the Department of Defense Language Ada; and
3. comparison of Pascal's logic and data structures with those of Basic, Cobol, Fortran, and PL/1.

PRECOLLEGE CAI

Diana Braiden Radspinner
Diann Dalton
Arlene Miller Smith
Janet M. Scott
Deborah C. Slaton
Katherine L. Helwick
Vicki S. Smith
Connie J. Brooks
M. Beatriz Beltran

ABSTRACT: Computer-Assisted Reading Skills

Diana Braiden Radspinner, Diann Dalton, Arlette Miller Smith, Instructional Technology Branch, Dallas Independent School District, 7131 Midbury, Dallas, TX 75230

The Computer-Assisted Reading Skills program is a developmental project that uses computer-assisted instruction for students in grades four through six. The program is aimed at supplementing and reinforcing those skills that have been assessed deficient. After identifying and categorizing the higher-order reading skills, the following were selected for development.

1. Identifying Fact and Opinion
2. Drawing Conclusions
3. Forming Generalizations

The microcomputer presents instruction, generates items for student response, reinforces correct answers and tracks student progress through the lesson. A student report is automatically displayed at the end of each learning session. The report provides the teacher with details of student performance. It includes a breakdown of student correct/incorrect responses to specific subskills within the lesson.

ABSTRACT: Dallas Microcomputer Mathematics Program

Janet M. Scott and Deborah C. Slaton, Instructional Technology Branch, Dallas Independent School District, 7131 Midbury, Dallas, TX 75230

The Dallas Microcomputer Mathematics K-8 Program (MCMP) is a guided drill and practice program designed to meet the needs of individual students by reinforcing and extending concepts presented by the teacher. MCMP comprises objective-based lessons in horizontal addition, vertical addition, horizontal subtraction, vertical subtraction, multiplication, division, fractions, decimals, numeration, ratio, proportion, and percent. Visual displays are included in initial skill lessons to enhance concept understanding. Each lesson contains guided instruction, positive reinforcement, and immediate assistance based upon the user's needs.

Student reports show the student's score, the number of problems attempted, the number correct on the first attempt, and the number correct on the second attempt. Data pertaining to subskills contained within each lesson are provided for more specific diagnosis and prescription by the teacher.

ABSTRACT: Computer-Assisted Mathematics Skills: Lesson Design and Development

Katherine L. Helwick and Deborah C. Slaton, Instructional Technology Branch, Dallas Independent School District, 7131 Midbury, Dallas, TX 75230

This Title IV-C project incorporates a systematic approach to designing instruction for microcomputer-based curriculum on mathematical problem solving skills. The project curriculum supplements and reinforces concepts and skills presented through regular classroom instruction in grades four through six. After identifying and categorizing problem solving skills, three areas were selected for lesson development: word problems, patterns, and sequences and series.

The lesson format provides optimum flexibility for diverse student needs. Each lesson contains a comprehensive

instructional sequence, an abbreviated instructional sequence (synopsis), and drill practice. Users can select any one or a combination of those lesson components. The microcomputer presents instruction, generates random problems, reinforces correct responses, provides correction and explanation of errors, and tracks student progress through the lesson. Performance reports provide general and detailed summaries of student progress.

ABSTRACT: Microcomputer Reading Program (MCRP)

Vicki S. Smith and Connie J. Brooks,
Instructional Technology Branch, Dallas
Independent School District, 7131 Midbury,
Dallas, TX 75230

Given the existing DISD Baseline of identified skills and current state-adopted texts, the Instructional Technology curriculum writers are developing a CAI reading program. The target population is those students enrolled in grades three through six who are reading one to two years below grade level and need remediation.

Using the Baseline levels as a guide, the following four strands were identified: (1) phonics, (2) structural analysis, (3) vocabulary, and (4) comprehension. Extensive research was conducted, classroom observations were made, and a systems approach was used to devise hierarchies for four levels.

The overall goal of this project is to produce a clean and interesting format to teach the basic skills outlined in the hierarchies, and the major instructional goal at each level is always a demonstration of the student's comprehension of the skills taught within the lessons. A systems approach is being used to clarify the task and ensure its successful completion.

ABSTRACT: Computer-Assisted Instruction Demonstration Project (ROLAR)

M. Beatriz Beltran, Dallas Independent
School District, Box 80, 3700 Ross Avenue,
Dallas, TX 75204

ROLAR was designed and is implemented to determine the effectiveness of a computer-based supplementary language system in improving the reading achievement level of Limited English Proficiency (LEP) students. The project uses the Bilingual Oral Language and Reading (BOLaR) and Region One Literacy Lessons (ROLL) curricula adapted for CAI. The supplemental language system reinforces classroom reading and language learning through individualized visual (screen display) and audio (voice synthesizer) language lessons in a transitional Spanish-English sequence. The project is housed at Gabe P. Allen Elementary School in Southwest Dallas. The CAI ROLL and BOLaR materials are being pilot tested with 120 first, second, and third grade students at G. P. Allen School and St. Mary of Carmel School.

ABSTRACT: Computer-Assisted Spanish English Transition Sequence

M. Beatriz Beltran, Dallas Independent
School District, Box 80, 3700 Ross Avenue,
Dallas, TX 75204

The CASETS program focuses on enhancing the English language skills of limited English Proficiency (LEP) students by integrating Social Studies-American History and the English Language Development course. Learning and transferring concepts and vocabulary from Spanish to English are approached through two support systems: computer-assisted instruction and native language materials.

The CAI activities and instructions, classroom activities and tapes for both the social studies and the English language development classrooms will be designed and pilot tested with seventh and eighth graders. The materials for eighth grade are now being design tested at one public middle school and one private school. The control group in the private school does not use CAI.

SOFTWARE DOCUMENTATION
FOR STUDENT PROJECTS

by
John A. Rohr
Computer Science Department
California State Polytechnic University, Pomona
Pomona, California

ABSTRACT

This paper describes the documentation normally required for a student project. Five documents are used: each successive document provides a more detailed description of the project than the previous document. The first is an information sheet about the student and the project idea; the second a short functional description of the project; the third a general design description including a preliminary user's guide; and the fourth a detailed design description complete with interface specifications. The fifth and final document is a complete project report.

INTRODUCTION

An essential component of any software project is adequate documentation. For a student project this may be the most important aspect. Without proper documentation the project reduces to a simple programming exercise rather than being a valuable learning experience.

Student projects usually consist of two types. The first type is a project done for the academic value, encompasses a subject of mutual interest to the student and the instructor, and the outcome is of interest mainly to the student. The second type of student project results in software that will be used by one or more persons other than the student doing the project.

For both types of projects, the students should learn proper documentation methods. For the second type, high-quality documentation must be provided for those who will use and maintain the software. For either type of project, the experience gained by the student in writing the documentation will be useful for his or her professional career.

DOCUMENTATION OVERVIEW

Adequate documentation for a student project consists of the following five documents:

1) Information Sheet
2) Functional Description
3) General Design Document
4) Detailed Design Document
5) Final Report

The Information Sheet is an administrative document that provides the instructor with personal and academic data about the student. The Information Sheet also contains a very brief description of the project.

The Functional Description is the complete specification of the project at a functional level. No implementation details are given. This document defines the scope of the project.

The General Design Document defines the structure of the software which will be generated. This document also includes a preliminary user's guide which gives operating instructions for the software.

The Detailed Design Document provides a complete description of the software. This document describes the general organization of the software and gives a detailed description of each module.

The Final Report is the most important document. This document gives a complete description of the entire project. It includes much of the material produced for the other four previous documents, but the Final Report presents all the information available about the project in a logical, organized, format.

INFORMATION SHEET
The Information Sheet provides the instructor with personal and academic data which may be useful during the course of the project. A short paragraph is included on the Information Sheet about the project. The personal information gives the instructor a means of contacting the student if that should be necessary. The academic data and the project description can be referred to during a conversation with the student to assess the student's ability to complete the project within the chosen time frame.

FUNCTIONAL DESCRIPTION
The Functional Description is the student's first complete, written specification of the project. This document should be concise but complete. On the average, one single-spaced typewritten page should suffice. This document should specify the computer system and the programming language which will be used. It should include an estimate of the human and computer resources which will be required, including any special resources such as tapes, disk packs, terminals, etc., which are not routinely provided to users of the computer system. Finally, a preliminary schedule for the entire project should be included.

GENERAL DESIGN DOCUMENT
The General Design Document specifies the internal organization of the software and the external interfaces with hardware, other software, and users. This document also includes a detailed schedule for the project. The first section of this document is the user's guide describing all the user interaction with the software including commands, input data, and output data.

Other sections of this document describe the global data base, the file formats, input-output formats, the preliminary module list, the top-level design, and the schedule for the project. The global data base includes all global data which are to be created. The type, range dimension, and all other pertinent attributes of each variable are given. Each file is described in terms of its organization and the structure of each type of record it can contain.

The module list and top-level design specify the structure of the software. Each module which can be identified at this stage is named and described briefly, and any segmentation for memory overlays is specified. Finally, a detailed schedule is given for the project showing when each document and program section will be available in preliminary and final form.

DETAILED DESIGN DOCUMENT
The Detailed Design Document describes the software as it is actually implemented. The document begins with an overview of the software. As design and coding proceed in a top-down manner, each module is designed, coded, and documented. The initial design for each module includes data specification and a flowchart or design-language description of the algorithm. When the module is completed, the design documentation is updated to reflect the actual implementation.
The description for each module includes data about the programmer and the program. The name of the module is given and the purpose of the module is explained briefly. Then the programmer's name, date written, language, and computer system are stated. Following are a list and description of local constants and variables, a list of global constants and variables, a list of procedures called, and a list of files used. File and input-output formats are then described. Finally a flowchart or design-language description of the module is given.

FINAL REPORT
The Final Report integrates all documentation produced for the project. The report is organized to present an increasing amount of detail as the reader progresses through the document. Much of the material for the Final Report is obtained from documentation produced earlier, although additional narrative material is required. A possible outline for a Final Report is as follows:

I. Introduction
II. Statement of the Problem
III. Overview of the Solution
IV. Details of the Solution
    A. External Interfaces
    B. Data Base
    C. File Formats
    D. Input-Output Formats
    E. Module Descriptions
V. Observations
Appendix: User's Guide

OBSERVATIONS
A student software project which includes adequate and proper documentation can be a valuable learning experience for the student. The documentation described in this paper can be as extensive or as brief as the instructor desires. All five documents should be required for all projects. In general, the larger the project,

the greater the amount of each of the documents.

The Information, Functional Description, and General Design Document should be completed before coding begins. The Detailed Design Document can be generated simultaneously with coding. The Final Report should be compiled after the coding, testing, and debugging are complete. Each document supercedes the previous document. No updating of previous documentation should be required until it is incorporated in the Final Report.

Good coding techniques should be encouraged. Comments should be used liberally. Each module should begin with its complete detailed description except for the flowcharts.

Finally, it is almost impossible to spend too much time documenting a project. Any time spent doing documentation is time well spent.

APPENDIX

DOCUMENTATION SUMMARY

- Information Sheet
- Functional Description (1 page)
    - Problem Description
    - Computer System
    - Programming Language
    - Special Resources
    - Preliminary Schedule
- General Design Document
    - Preliminary User's Guide
    - Global Data Base
    - File Formats
    - Input-Output Formats
    - Top-Level Design
    - Module List
    - Schedule
- Detailed Design Document
    - Program Structure
    - Module Descriptions
        - Name
        - Purpose
        - Author
        - Date Written
        - Computer System
        - Programming Language
        - Local Constants
        - Local Variables
        - Global Constants
        - Global Variables
        - Procedures Called
        - Files Used
        - File Formats
        - Input-Output Formats
        - Algorithm Description
            - Flowchart or
            - Design-Language Description
- Final Report
    - Introduction
    - Statement of the Problem
    - Overview of the Solution
    - Details of the Solution
        - External Interfaces
        - Data Base
        - File Formats
        - Input-Output Formats
        - Module Descriptions
    - Observations
    - User's Guide

FIRMWARE DEVELOPMENT

Robert N. Cook
Computer Science Department
Central Michigan University
Mt. Pleasant, MI 48859

INTRODUCTION

Traditionally, computer science departments have taught high-level and low-level language assembly language courses. Computer design courses often cover mainly hardwired control units where the instruction set was designed using gates. Recently the availability of cheap read--only memory (ROM) has lead to the widespread use of microprogrammed control units, where a microroutine implements each machine language instruction. These microroutines are neither hardware nor software in the traditional sense, hence the name "firmware" as coined by Ascher Opler in the January 1967 issue of Datamation.

At Central Michigan University microprogramming has been used in CPS 560-Digital Computer Design where the students design a small general purpose microprogrammable digital computer previously described by Vishnubhotlas (9,10). In the undergraduate course, CPS 360--Logic Circuit Design, which all majors are required to take, microprogrammable control units are introduced using a simple hypothetical computer. For the benefit of readers not familiar with microprogramming, this computer is explained in the next section of this paper. In addition to these courses, a graduate level special topics course on microprogramming has been taught. This course used translator and simulator programs written in Fortran which allowed the students to write and execute microprograms for the Burroughs D-machine.

Recently the Computer Science Department of Central Michigan University acquired a Burroughs B1830 which is user microprogrammable using Micro Implementation Language (MIL)(2). With software package available from the University of Utah, students can write microprograms using an extended version of MIL (McMIL) and execute them (6).

The primary application of microprogramming is emulation, where the instruction set of a guest computer is microprogrammed on a host computer. The host computer is then said to emulate the guest computer. Thus programs written for the guest computer will be executable on the host computer without conversion. By designing a family of computers all of which are microprogrammable, all computer models of the family may emulate the same system architecture. Then programs written for any model of the family will run without conversion on any other model. The marketing advantages to the computer manufacturer and the lack of conversion problems for the user when changing to a different model computer provide strong incentives to use this approach to designing a family of computers.

This paper also discusses an elementary digital computer (EDC) that is defined(4). This computer's instruction set is sufficiently complex to serve as the project for a 3 semester hour course in firmware development. The EDC is emulated on the user microprogrammable host computer (Burroughs B1830 for example) or the microprograms may be written for the Burroughs D-machine using the translator and simulator programs described by Katzan(5). Recently, in fact, one of our M.S. candidates used these programs to emulate the IBM 1130 computer (see Shyu(8)).

This paper shows how firmware development can be integrated into computer science undergraduate curricula. A simplified microprogrammable machine suitable for a first course in logic design is detailed. Further, an elementary digital computer is defined which is suitable for emulation in a three semester hour firmware development course.

A SIMPLIFIED MICROPROGRAMMABLE COMPUTER

The machine described in this section is a modified version of the machine described by Bartee (1), the current text in CPS 360-Logic Circuit Design. The entire course is described by Cook in (3). An alternative hypothetical microprogrammable machine is described in chapter 1 of Katzan (5). Figure 1 shows the register configuration of the simplified microprogrammable computer:

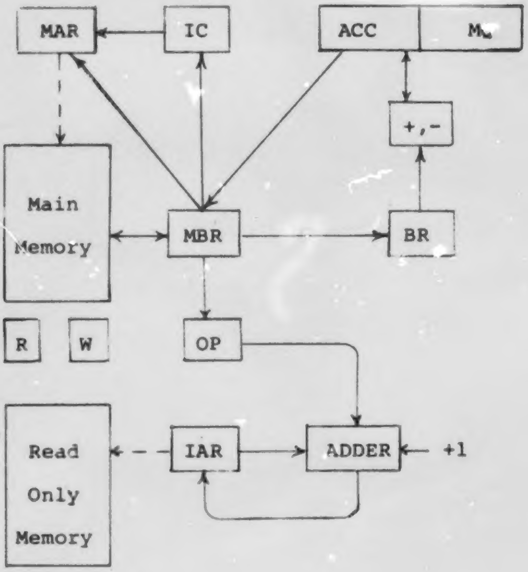


Figure 1
A Simplified Microprogrammable Computer

Registers used in this machine are:

MBR	Memory Buffer Register
MAR	Memory Adder Register
OP	Operation Code Register
BR	Arithmetic Input Register
ACC	Accumulator
MQ	Accumulator Extension
IC	Instruction Counter
R	Read Flip Flop
W	Write Flip Flop
IAR	Instruction Address Register
ADDER	IAR ← OP + IAR + 1
+,-	ACC ← ACC + B, ACC ← ACC - B performed here

Fundamental Micro operations available on this machine are:

MICROOPERATION	ACTION
R ← 1	MBR ← (Memory) (Read)
R ← 0	Reset Read flip flop
W ← 1	(Memory)$_{MAR}$ ← (MBR) (Write)
W ← 0	Reset Write flip flop
OP ← MBR	Op code portion of MBR to OP register
IC ← MBR	Address portion of MBR to Instruction Counter
IC ← 0	Zero Instruction Counter
MAR ← MBR	Address portion of MBR to Memory Address Register
IC ← MAR	MAR to Instruction Counter
BR ← MBR	Memory Buffer Register to B Register
ACC ← ACC+B	Add
ACC ← ACC-B	Subtract
ACC ← 0	Zero Accumulator
IC ← IC+1	Increment Instruction Counter
MBR ← ACC	Accumulator to Memory Buffer Register
IAR ← 0	Instruction Address Register addresses word 0 of ROM
IAR ← IAR+1	Increment IAR
IAR ← OP+IAR+1	Compute Branch Table Address
IAR ← N	Absolute address in ROM to Instruction Address Register

To students, the most confusing part of this machine is the Instruction Address Register. Just as the instruction counter (IC) holds the address of the next machine language instruction in main memory, the IAR holds the address of the next microinstruction to be executed in the ROM. Thus each microinstruction must modify the IAR to locate the next microinstruction that will be executed.

Assuming that the op codes of the machine language are assigned sequentially beginning with zero, the correct microroutine to implement a given machine language instruction may be found using these microinstructions:

ROM LOCATION	MICROOPERATIONS	ACTION
0	R ←1, IAR ←IAR+1	Read machine language instruction to Memory Buffer Register
1	OP ←MBR, IAR←IAR+1	Op code to OP register
2	IAR ←OP+IAR+1	IAR ←OP+3
3	IAR ←50	Branch to ROM_{50}
4	IAR ←55	Branch to ROM_{55}
5	IAR ←60	Branch to ROM_{60}
6	IAR ←64	Branch to ROM_{64}
⋮		
50		Microroutine for Op code 0
⋮		
55		Microroutine for Op code 1
⋮		
60		Microroutine for Op code 2
⋮		
64		Microroutine for Op code 3
⋮		

The microinstruction in word 0 of ROM reads the machine language instruction into the MBR and increments the IAR to 1. In word 1 the microinstruction moves the op code portion of the machine language instruction to OP register and increments the IAR to 2. Word 2 of ROM changes the IAR to IAR+OP+1. Since IAR=2 the effect is IAR←OP+3. Then, after word 2, control is transferred to ROM 3,4,5,6,... If the op code is 0,1,2,3.... The branch table (jump table) in words 3,4,5,6... of ROM then transfers control to words 50,55, 60,64... of ROM where the microroutines to implement the op codes 0,1,2,3... are found. The sequence of execution of microinstructions is then:

OP CODE	SEQUENCE OF EXECUTION IN ROM
0	0,1,2,3,50...
1	0,1,2,4,55...
2	0,1,2,5,60...
3	0,1,2,6,64...
⋮	⋮

As an example, consider the microroutine to implement the ADD machine language instruction. If the op code is 2 the microroutine begins in ROM location 60:

ROM LOCATION	MICROOPERATIONS	ACTION
60	MAR ←MBR	Data Address To MAR
	IAR ←IAR+1	Increment IAR
61	R ←1,IC ←IC+1 IAR ←IAR+1	Read Data To MBR Increment IAR
62	R ←0,BR -MBR	Data to B Register
	IAR ←IAR+1	Increment IAR
63	ACC ←ACC+B MAR ←IC	Add Address of next Instruction to MAR
	IAR ←0	Branch to word 0 of ROM

Initially the machine language instruction is in the Memory Buffer Register because of the read that was performed by word 0 of ROM. Word 60 moves the address portion of the Instruction to the Memory Address Register so that the data to be added can be read by word 61. In addition, word 61 increments the Instruction Counter to point to the next machine language instruction in main memory. Word 62 resets the R flip flop and moves the data item to the B Register so that it may be added to the accumulator by word 63. Each of the above microinstructions also increments the Instruction Address Register to address the next word of ROM. In word 63 the Instruction Counter is moved to the Memory Address Register so that ROM word 0 will read the next machine language instruction. Additionally the instruction Address Register is set to 0 to begin the execution of the next machine language instruction.

The subtract microprogram is the same as the add microprogram except that ROM word 63 is changed to:

ACC ←ACC-B,
MAR ←IC,
IAR ←0

For the load (clear and add) microroutine only the instruction ACC←0 must be added to ROM word 62. The subtract and load microroutines would, of course, be located in the appropriate ROM locations (not 60-63).

More complex machine language instructions such as conditional branches and multiply instructions can be implemented using other microoperations. For example, if the microoperation

IF(ACC_0=1) THEN IAR ← N ELSE IAR ← IRA+1

is added to the fundamental operations described earlier, then the Branch on Results Minus microroutine may be written:

ROM LOCATION	MICROOPERATIONS	ACTION
75	IF(ACC_0=1)	
	THEN IAR ← 77 ELSE IAR ← IRA+1	-:Branch to 77 +:Branch to 76
76	IC ← IC+1 IAR ← 78	Increment Instruction Counter
77	IC ← MBR IAR ← IAR+1	Address portion of MBR to Instruction Counter
78	MAR ← IC IAR ← 0	Address of next Machine language Instruction to MAR

In this microroutine, the Instruction Counter is incremented (76) if the ACCumulator is positive. The address portion of the instruction, which gives the main memory address to branch to, is moved to the Instruction Counter from the Memory Buffer Register (77) when the ACCumulator result is negative. After the Instruction Counter is altered by microinstruction 76 or 77, then microinstruction 78 puts this address into the Memory Address Register and sets the Instruction Address Register to 0 to read the next machine language instruction from main memory. Both this BRM microroutine and the multiply microroutine, which requires several additional microoperations not discussed here, are used as homework problems in CPS 360.

All these microroutines assume that the machine language being implemented uses direct addressing only. This restriction may be removed, if desired, in a separate course in firmware development. As the study of microprogrammable control units is but one part of an already overcrowded course, the direct addressing restriction is needed here.

AN ELEMENTARY DIGITAL COMPUTER

The instruction set of an elementary digital computer (EDC) presented in this section is suitable for emulation in a three semester hour course in firmware development. A significant amount of the time must be devoted to the architecture of the host computer which will emulate the EDC and to the syntax of the microprogramming language used. Either an actual computer which is user microprogrammable such as the Burrough B1830 or the transistor and simulator Fortran programs for the Burrough D-machine may be used to emulate the EDC.

An overview of the architecture of several microprogrammable computers is given by Salisbury (7). For schools which have available a Burroughs computer of the B1700/B1800 series. Organic (6) describes the series architecture as well as the microprogramming language MIL and its extension McMIL. In this book the SAMOS computer is emulated in McMIL. Rather than emulating a separate computer such as the EDC described here, it is entirely reasonable to extend and enhance the SAMOS interpreter given by Organic (see Appendix E of reference 6).

The EDC is a simple, single-address von Neumann type digital computer using two's complement representation of numbers. Memory consists of 1 K(1024) 16-bit words. Thus the ten-bit address field can directly address all available memory. The remaining six bits of the instruction contain four bits for the OP code field and two bits for the IX field. This IX field indicates the addressing mode or extends the OP code field to indicate up to four similar instructions with the same OP code bits (for example the four transfer instructions).

Figure 2 shows the register configuaration of the EDC.

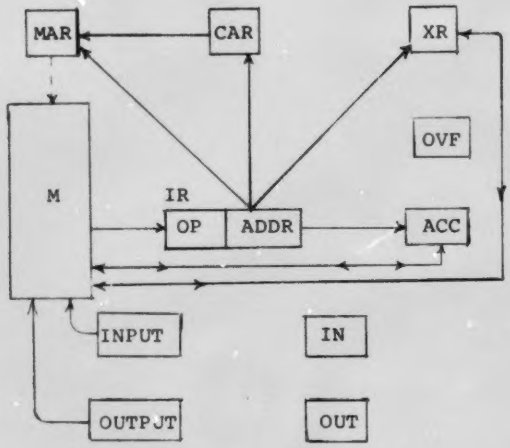


Figure 2
The Elementary Digital Computer

Components of the EDC are:

M	Main memory - 1024 16 bit words
IR	Instruction Register containing OP and ADDR subregisters
OP	OP code bits of IR
ADDR	ADDRess bits of IR
CAR	Current Address Register
MAR	Memory Address Register
ACC	Accumulator
XR	Index Register
INPUT	INPUT register which buffers the input device
IN	Input flip flop
OUTPUT	OUTPUT register which buffers the output device
OUT	Output flip flop
OVF	Overflow flip flop

An instruction set for this computer is described below. The OP co-e bits and IX bits, if appropriate, are shown in parenthesis after the mneumonic OP code.

INP (0000)

This instruction moves one word from the INPUT buffer register to main memory at the address specified by the ADDR subregister. Flip flop IN is set by the input equipment when the buffer has been loaded and reset by the INP microroutine after the word is transferred to main memory. Thus flip flop IN provides the necessary two-way communication link between the control unit and the input equipment.

OUT(0001)

The OUT microroutine uses the OUTPUT buffer register and the OUT flip flop to transfer a main memory word to the output equipment.

BIN (0010)

As the INP microroutine moves one word from the INPUT buffer register to main memory, the Block INput instruction moves N words from the INPUT buffer register into main memory beginning at the address originally in the ADDR subregister. The ACCumulator is previously loaded with N and then decremented by one until all N words are transferred. At the same time the Memory Address Register is incremented by one to provide the correct addresses in main memory.

ZRO(0011)(00), ADV(0011)(01),
DBL(0011)(10), CSN(0011)(11)

These four instructions modify the contents of the ACCumulator. ZRO clears the ACC to all zeros. ADV adds one to the ACC. DBL doubles the ACC by adding it to itself. CSN takes the two's complement of the ACC including the sign bit.

LDA (0100)

The ACC is loaded from main memory at the address given by the ADDR subregister (the address portion of the instruction).

ADD(0101), SUB(0110)

Here the contents of the memory location specified by the ADDR subregister are added to or subtracted from the contents of the ACC and the result is left in the ACC.

STO (0111)

The STO microroutine stores the contents of the ACC in main memory at the location specified by the ADDR subregister.

AND (1000)

To provide masking for ACC results, the AND microroutine performs the bit by bit "and" of the ACC and the main memory word specified by the ADDR subregister.

TRA(1001)(00), TRZ(1001)(01),
TRN(1001)(10), TRO(1001)(11)

Four transfer instructions provide unconditional transfer (TRA), transfer on zero (TRZ), transfer on negative (TRN), and transfer on overflow (TRO). These instructions load the Current Address Register from the ADDR subregister if the ACC condition specified is true, otherwise the CAR is incremented by one and the transfer does not take place.

LDI(1010)(00), INC(1010)(01)

Both of these instructions are examples of immediate addressing where the ADDR subregister holds actual data, not the address of the data in main memory. As the ADDR subregister is only ten bits long while the ACC is 16 bits long, the LDI instruction loads the ACC from the ADDR subregister with the leftmost bit duplicated six times. Thus the two's complement representation of the shorter integer in the ADDR subregister is preserved when it is loaded into the ACC. The INC microprogram increments the ACC by adding the contents of the ADDR subregister.

SRA(1011)(00), SRL(1011)(01),
CIR(1011)(10)

Other immediate addressing instructions are the shift right arithmetic (SRA), shift right logical (SRL), and the circular right shift (CIR). Here the number of bits to be shifted is specified by the ADDR subregister. The SRA instruction extends the sign bit when shifting and the SRL uses zero fill.

BEX (1011)(11)

To facilitate character manipulation, the BEX microroutine exchanges the left and right ACC bytes.

In the discussion of all of the instructions above either direct or immediate addressing was assumed. LDI, INC, SRA, SRL, and CIR used immediate addressing and the others use direct addressing. Depending on the level of the course and the availability of time to devote to the project, the emulation may stop after the instructions described so far have been microprogrammed. If desired, up to three additional addressing modes may be incorporated for those instructions not already using the IX bits. A scheme for interpreting the IX bits is:

IX	Addressing Mode
00	Direct
01	Relative
10	Indexed
11	Indirect

In relative addressing the contents of the address portion of the instruction (ADDR subregister) are added to the Current Address Register to calculate the effective address.

With indexed addressing the effective address is the sum of the ADDR subregister and the index Register. If the XR is included in the EDC, three additional instructions must be microprogrammed.

LXR(1100)

Loading the index register is done, using one of the four addressing modes specified by the IX bits, by the Load index Register microroutine.

SXR (1101)

The contents of the index register can be stored in main memory at the effective address specified by any of the four addressing modes using the Store index Register instruction.

DBX (1110)

Looping is accomplished in the EDC's machine language using the Decrement and Branch on index register zero instruction. Here the index register is decremented by one and compared with zero. If the index register is greater than zero, then the branch to the address given by the address portion of the instruction (ADDR subregister) takes place, otherwise the next sequential machine language instruction is executed to terminate the loop. This instruction is ideal for a post-test loop. If pretest loops are desired, this instruction could be modified, or the one remaining unused OP code combination (1111) may be used to define an additional looping instruction.

Indirect addressing (IX=11) requires no additional instructions to be defined. In this addressing mode the ADDR subregister contains not the address of the data but rather the address of the memory location which contains the a-dress of the data. If the memory is to be expanded to more than the 1 K words defined previously, the ten bit address field is insufficient to directly address all of memory. With indirect addressing all 16 bits are available for a memory address and up to 64 K (65,536) words may be addressed.

The additional addressing modes may be implemented using a single microroutine which is called only by the microroutines for those machine language instructions which allow the four addressing modes. Some instructions use the IX bits to extend the OP code field rather than to determine addressing mode. Therefore this approach is preferable to having the fetch microroutine (which reads the next machine language instruction, decodes the instruction, and branches to the correct microroutine to execute the instruction) compute the effective address for all of the instructions.

In addition to the fetch microroutine, the microroutine which computes the effective address of the data, and the microroutines for the various OP codes described previously, an error microroutine is required to handle the invalid OP code (1111 is the only invalid OP code in the EDC as defined above). The microroutine which computes the effective address should also produce an error message if an invalid address is computed.

Along with the syntax of a microprogramming language and the architecture of the host computer, the emulation of the elementary digital computer described in this section provides entirely enough material for a 3 semester hour course in firmware development.

SUMMARY

This paper has described two means by which firmware development can be integrated into an undergraduate computer science program. In the first course in logic design, a microprogrammable control unit is discussed. Here the use of Read Only Memory to store microinstructions is discussed. Simple microroutines are written and the branch table concept along with the register configuration of a simplified microprogrammable computer are given. The concept of emulation where a computer is microprogrammed to emulate a specified architecture is introduced.

To extend these concepts a three semester hour course in firmware development is discussed. In this course the syntax of a microprogramming language for a user microprogrammable computer such as the Burroughs B1700/B1800 series is taught. If such a machine is not available, translator and simulator programs for the Burroughs D-machine written in Fortran may be used. The project for the firmware development course is the emulation of an elementary digital computer. Twenty-two instructions for the EDC are described. If indexed addressing is to be included, 3 additional instructions are defined. Addressing modes may be limited to direct and immediate addressing or extended to include relative, indexed, and indirect addressing.

Because of the increased number of computers which are microprogrammed at the factory and user microprogrammable computers which are used as emulators, the employment prospects of our graduates who have a background in firmware development should indeed be excellent.

REFERENCES

1. Bartee, Thomas C., "Digital Computer Fundamentals - Fifth Edition", McGraw-Hill (1981).
2. Burroughs Corporation, "B1700 Systems Micro Implementation Language (MIL) Reference Manual", Detroit, MI (1977).
3. Cook, Robert N., "A Hardware Course For A Software Curriculum", ACM SIGCSE Bulletin (to appear).
4. Dietmeyer, Donald L., "Logic Design of Digital Systems-Second Edition", Allyn and Bacon (1979).
5. Katzan, Harry Jr., "Microprogramming Primer", McGraw-Hill (1977).
6. Organic, Elliot I. and Hinds, James A., "Interpreting Machines: Architecture and Programming of the B1700/B1800 Series", Elsevier North-Holland (1978).
7. Salisbury, Alan B., "Microprogrammable Computer Architectures", American Elsevier (1976).
8. Shyu, Wen-Chang, "Implementation of Microprogramming", M.S. Plan B Paper, Central Michigan University, Computer Science Department (1980).

9. Vishnubhotla, Sarma R., "A Logical Approach To Teach Computer Design At Logic and System Level", ACM SIGCSE Bulletin Vol. 9, No. 1 (1977).

10. Vishnubhotla, Sarma R., "A Project To Teach Microprogrammed Asynchronous System Design", ACM SIGCSE Bulletin Vol. 10, No. 1 (1978).

Cognition and Programming: Why Your Students Write Those Crazy Programs

Elliot Soloway, Jeff Bonar, Beverly Woolf,
Paul Barth, Eric Rubin, and Kate Ehrlich

Computer and Information Science Department
University of Massachusetts
Amherst, Massachusetts 01003

Abstract

In this paper, we first argue that there is a great need for empirical research coupled with carefully articulated theories. Psychological claims of language designers and advocates call out for evaluation. Moreover, computing education, to rise above instruction in only the syntax and semantics of language constructs, needs a description of the knowledge programming experts know and use. We then present a network which represents the high-level, plan knowledge an expert may possess with respect to aspects of looping and assignment. Based on this knowledge, we look at actual student programs, and attempt to understand the possible misconceptions students had, which manifested themselves as buggy programs. Finally, we make suggestions for computing education which reflect the insights gained in developing this knowledge network and in the analysis of the buggy programs.

I. Introduction

I.1 Introductory Polemic

Designers and advocates of programming languages are continually making claims that their languages are simple, readable, encourage better programming, or encourage natural problem solving habits.

> The development of the language Pascal is based on two principle aims. The first is to make available a language suitable to teach programming as a systematic discipline based on certain fundamental concepts clearly and naturally reflected by the language. ... The syntax of Pascal has been kept as simple as possible. ... This property facilitates both the understanding of programs by human readers and the processing by computers.
>
> Wirth [1977]

> If ordinary persons are to use a computer, there must be simple computer languages for them. BASIC caters to this need ....
>
> Kurtz [1978]

> Happily, LISP is easy to learn. A few hours of study is enough to understand some amazing programs. ..... One reason LISP is so easy to learn is that its syntax is extremely simple.
>
> Winston [1981]

This work was supported by the Army Research Institute for the Behavioral and Social Sciences, under ARI Grant No. MDA903-80-C-0508.

Any opinions, findings, conclusions or recommendations expressed in this report are those of the authors, and do not necessarily reflect the views of the U.S. Government.

> APL is one of the most concise, consistant, and powerful programming languages ever devised.
>
> McCracken [1976]

Notice that the above claims are psychological claims and thus are open to empirical inquiry. Where, then, is the empirical research backing up these claims? Moreover, it is not enough to have data -- one needs theories which explain and account for the data. What are needed, then, are carefully articulated theories which suggest why program construct x should be more readable, useful, etc. than program construct y?

Moreover, in the rush to claim readability, etc. for their particular programming language, designers have often forgotten a key aspect of cognition: learning. Can individuals learn to understand program construct x? How easily? The extensive use of Pascal, Basic, Lisp, and APL does not negate the relevance of the learnability question: humans can adapt to most any environment. Thus, no matter how difficult a particular language might be to learn, many individuals can and do persevere to conquer it. But at what cost? And what about the ones that don't succeed? Do we want to simply write them off? As we all know, with the coming of the computer age, everyone will need to know how to program to some degree.

The picture is not as bleak as we've painted it. Fortunately, studies have been and are being done, which attempt to understand the relationship between cognition and programming. For example, Ledgard, et. al. [1980] found that a command language which has a natural language structure, as opposed to the typically baroque constructs, facilitates easier and more effective use by humans. Welty and Stemple [1981] compared the performance of novice users with a procedural and non-procedural query langauge and found that novices using the procedural language did better when formulating moderate to difficult queries. Dunsmore and Gannon [1978] explored the factors that contribute to program complexity, while Gannon [1978] catalogued bugs in student programs. L. Miller [1978] has explored a whole range of behavioral issues in programming. Among other issues, Meyer [1980] has explored the use of concrete models in programming education. Finally, Shneiderman [1980] contains a distillation of issues in, as he puts it, software psychology.

A word of caution: this type of research is difficult to carry out. Experiments can always be criticized; theoretical interpretations can always be faulted. Those in this paper are no exception. Moreover, the climate in computer science has not always been receptive to, or even tolerant of, this type of work. The need to study cognitive factors in computing, however, grows ever more apparent. Thus, we feel it imperative that such research be encouraged, and the support mechanisms to insure its existence arise.

I.2 Substantive Introduction

This paper will serve as a progress report on the current work of the Meno Research Group at UMass. The goals of our project are twofold. First, we are building Meno-II, a run-time support environment for novice Pascal users; this system will catch run-time errors (not compile-time errors [1]), and help the student debug the program -- and his/her understanding -- by providing tutoring in the areas underlying the errors. The other, complementary goal of this project is to understand:

1. what one knows when one solves problems of the sort used in introductory Pascal courses,
2. what program bugs are typically made by novice programmers,
3. how knowing about what should be understood can be used to explain the mind bugs (misconceptions) students have which result in program bugs.

In this paper, we describe our efforts to date on this latter goal.

---

[1] We are not concentrating on compile-time errors, since systems which cope with such errors already exist. For example, Teitelbaum [1980] has built a programming environment which helps eliminate syntax errors by not allowing programmers even to enter ill-formed structures.

The perspective we have taken in this research is based on the following observation: as experts, when we write programs to solve problems, we use lots of high-level knowledge. Most of this knowledge is much deeper than the syntax and semantics of any given programming language. In particular, an expert knows how to decompose problems into fairly standard tasks, e.g., accumulate a running total and search through the elements of this array. This knowledge is highly organized into bundles, or frames [Minsky, 1975], and the rich fabric of bundles are knit together by relationships. The key role which this tacit knowledge plays in the problem solving behavior of experts in a wide variety of areas is gaining increasing attention [Collins, 1978]. The difficult problem is to ferret out this knowledge and make it explicit, much as Dijkstra [1976], Wirth [1977], and Friedman [1974] have attempted to do in their books on programming. Thus, it is this knowledge, rather than just the syntax and semantics of the particular programming language, that needs to be taught to the growing numbers of computing students.

In the following section we present a first-pass at describing the knowledge an expert might know about aspects of looping and assignment. Section III concentrates on how this knowledge can be used to predict and explain students' bugs. There we examine data -- bugs -- collected from a test we administered to introductory Pascal programmers. In Section IV we summarize the key theses of the paper.

II. Fragments of an Expert's Knowledge: The Theory

We commented above that an expert programmer knows about structures that help him/her relate *problems* to *programs*. These intermediate entities reflect a process of abstraction and condensation. That is, problems, and their solutions, are grouped together based on various criteria of similarity and their salient characteristics are highlighted. When confronted with an ostensibly new problem, the expert calls upon these structures and tries to find an old problem (and solution) which is similar to the new problem. Solving the new problem, then, is based on modifying the solution to the old problem. These intermediate structures are crucial for this type of problem solving (Papert, [1980]; Rissland and Soloway, [1980]).

While others have described the knowledge possessed by experts in mathematics (e.g., Polya [1963], Rissland [1980]) and physics (e.g., Larkin, et al. [1980]), we have attempted here to articulate the knowledge a programming expert might have with respect to a few problem types and a few related program constructs. We have borrowed from artificial intelligence a representation of knowledge, called frames, in which to encode our efforts. One key idea behind this particular knowledge representation -- and one consistent with our view of what an expert might know -- is that concepts are not necessarily atomic entities; rather they can have internal structure, such as descriptor types and descriptor values. Thus, a frame represents a template or boilerplate with slots which can be filled in. Finally, frames are tied together by different types of relationships, e.g., A-KIND-OF, or USES. [1]

Figure 1 depicts a fragment of an expert's knowledge -- plans -- encoded as frames. Consider, in particular, the Running_Total Plan Frame. This structure reflects the observation that many problems, especially ones used in introductory courses, require accumulating a total (e.g., the sum or product of the first n integers). Thus, given the ubiquity of this problem type an expert might explicitly represent it in his/her memory. From Figure 1, we also see that this frame is A-KIND-OF Loop Plan, i.e., a specialization of the general notion of a loop plan, and further that the Counter Loop Plan Frame is a specialization of Running Total Loop Plan. We see that various specific Pascal loop constructs (*repeat*, *while*, *for*) are associated with this plan. Finally, we see that two variable frames, the Running_total Variable and the New_Value Variable are used to describe the Running Total Loop

---

[1] In this discussion, we have glossed over many technical distinctions. However, for our purposes, the commonsense notion of "plan" and "frame" are sufficient. The interested reader can follow these issues more carefully in texts such as Nilsson [1980].

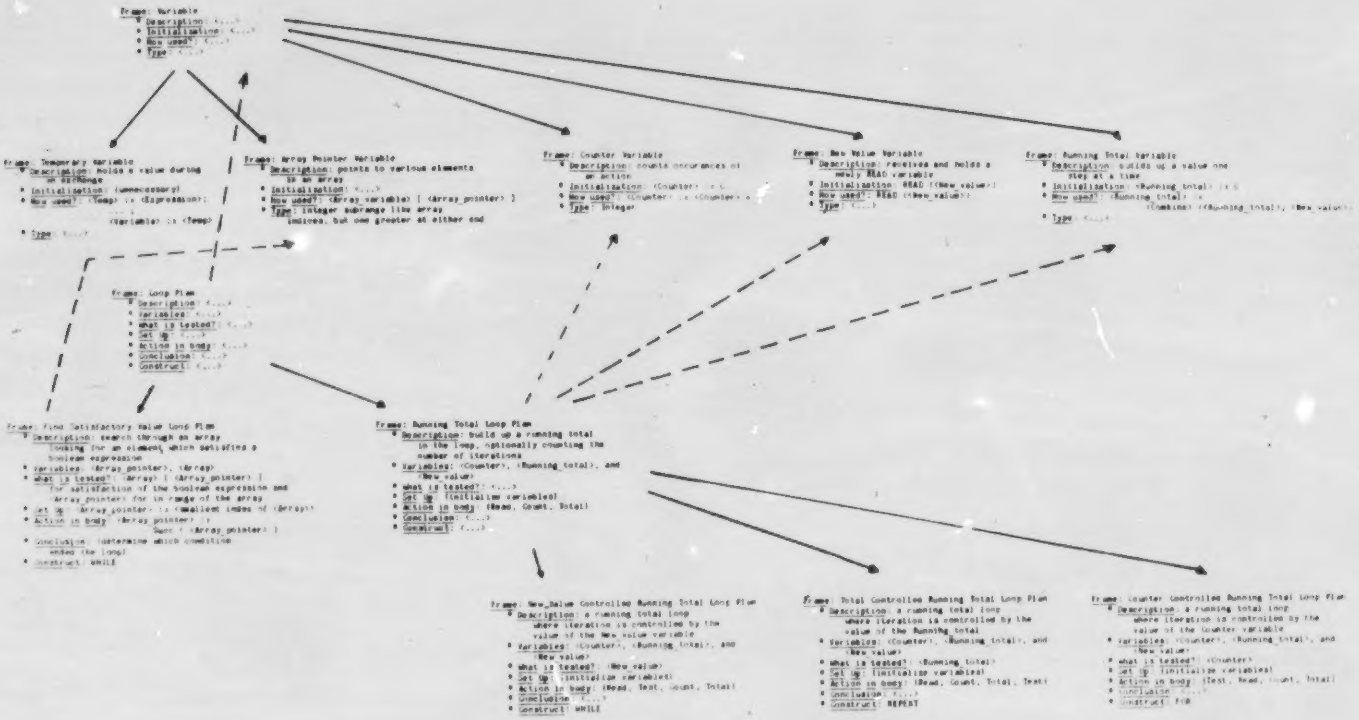

Frame: Variable
Description: <...>
Initialization: <...>
How used?: <...>
Type: <...>
Frame: Temporary Variable
Description: holds a value during an exchange
Initialization: (unnecessary)
How used?: <Temp> := <Expression>;
... ;
<Variable> := <Temp>
Type: <...>
Frame: Array Pointer Variable
Description: points to various elements in an array
Initialization: <...>
How used?: <Array_variable> [<Array_pointer>]
Type: integer subrange like array indices, but one greater at either end
Frame: Counter Variable
Description: counts occurances of an action
Initialization: <Counter> := 0
How used?: <Counter> := <Counter> + 1
Type: Integer
Frame: New Value Variable
Description: receives and holds a newly READ variable
Initialization: READ (<New_value>)
How used?: READ (<New_value>)
Type: <...>
Frame: Running Total Variable
Description: builds up a value one step at a time
Initialization: <Running_total> := 0
How used?: <Running_total> := <Combine> (<Running_total>, <New_value>)
Type: <...>
Frame: Loop Plan
Description: <...>
Variables: <...>
What is tested?: <...>
Set Up: <...>
Action in body: <...>
Conclusion: <...>
Construct: <...>
Frame: Find Satisfactory Value Loop Plan
Description: search through an array looking for an element which satisfies a boolean expression
Variables: <Array_pointer>, <Array>
What is tested?: <Array> [<Array_pointer>] for satisfaction of the boolean expression and <Array_pointer> for in range of the array
Set Up: <Array_pointer> := <smallest index of <Array>>
Action in body: <Array_pointer> := Succ (<Array_pointer>)
Conclusion: (determine which condition ended the loop)
Construct: WHILE
Frame: Running Total Loop Plan
Description: build up a running total in the loop, optionally counting the number of iterations
Variables: <Counter>, <Running_total>, and <New_value>
What is tested?: <...>
Set Up: (initialize variables)
Action in body: (Read, Count, Total)
Conclusion: <...>
Construct: <...>
Frame: New_Value Controlled Running Total Loop Plan
Description: a running total loop where iteration is controlled by the value of the New value variable
Variables: <Counter>, <Running_total>, and <New_value>
What is tested?: <New_value>
Set Up: (initialize variables)
Action in body: (Read, Test, Count, Total)
Conclusion: <...>
Construct: WHILE
Frame: Total Controlled Running Total Loop Plan
Description: a running total loop where iteration is controlled by the value of the Running total
Variables: <Counter>, <Running_total>, and <New_value>
What is tested?: <Running_total>
Set Up: (initialize variables)
Action in body: (Read, Count, Total, Test)
Conclusion: <...>
Construct: REPEAT
Frame: Counter Controlled Running Total Loop Plan
Description: a running total loop where iteration is controlled by the value of the Counter variable
Variables: <Counter>, <Running_total>, and <New_value>
What is tested?: <Counter>
Set Up: (initialize variables)
Action in body: (Test, Read, Count, Total)
Conclusion: <...>
Construct: FOR

Figure 1. A network representing a theory of simple looping programs in Pascal. The "Frame"s represent chunks of knowledge available to experienced programmers. These chunks are held together by "A-Kind-of" arcs (solid lines) and "Uses" arcs (dashed lines). Items in brackets are descriptions of things that must be inserted when the Frames are used. Standard Pascal identifiers are in all capitals.

Plan.

From a syntax and semantics view, the distinctions made in Figure 1 would disappear. For example, the Running_Total Variable and the Counter Variable are the same at those levels. However, at the functional, pragmatic level we see differences in the roles which variables and constructs play. Figure 1 highlights those functional differences. Moreover, as we shall see in our discussion of program bugs (next section), students performed differently on constructs, which on the syntax and semantics level, are the same.

We do not mean to suggest that the knowledge incorporated into Figure 1 is the final word. In fact, other experts have proposed alternative schemes (e.g., Waters [1979]; Soloway and Woolf [1980]; Miller, M. [1978]; Goldstein [1974]). Regardless of the details, experts do have and use structures similar to those in Figure 1, and we should be teaching students based on this view.

Besides using such knowledge for teaching, we see another important use for it. It can help us to understand bugs, program bugs and mind bugs, committed by novice programmers. First, we subscribe to the theory of learning that states that students understand something by constructing their own knowledge frameworks. A mind bug, then, is a "faulty" knowledge net, when compared with an expert's. That is, "bug" is only a bug because it does not agree with some standard. Moreover, a good teacher, who also has a good knowledge net, can use the student's bug as a window into the student's knowledge net. Brown and Burton [1978] use this approach in their analysis of students' buggy subtraction algorithms. Remediation can then be developed which is tuned specifically to the subtleties of the student's buggy model.

In the next major section of this paper, we shall use aspects of the knowledge in Figure 1 to analyze program bugs we collected from novice programmers. We found that *without* this type of high-level knowledge, many of the buggy programs seemed anomalous at best, but with the development of knowledge of the sort depicted in Figure 1, these bizarre programs became understandable to a large degree.

III. Understanding Program Bugs and Mind Bugs: The Data

III.1 The Context of the Test and the Test Instrument

Table 1 contains three of the problems we asked 31 introductory Pascal programming students to solve. The course was given in the summer session of 1980; the test was administered on the last day of classes. The students ranged from freshman to graduate students. Table 1 depicts the overall performance of students on these questions; the percentages are surprisingly low, given the ostensibly simple nature of the problems and the advanced state of the class. Even if one ignores what might be called easily detectable mistakes (e.g., no initialization of a variable), the percentages do not rise appreciably.

A reason for this low performance, and an important criticism of programming tests given where students do not have access to a computer is that introductory programming students almost never write a program correctly right off the bat. Rather, they go to the terminal and, over possibly several sessions, evolve a correctly running program.
Nonetheless, data gathered from written tests does suggest trouble spots, which if resolved might lead to more effective problem solution/program development. [1]

---

[1] We are also collecting on-line protocols; as a student sits at a terminal and submits a run to the Pascal system, our system makes a copy of the student's program and saves it. Thus, we have a record of the student's interactions. So far, we have data on 3 introductory Pascal classes. We are analyzing these data now. We also interview students individually, recording these sessions on audio tape (we hope to soon go to video tape). These type of data is particularly illuminating; whereas the on-line data and the test data are simply records of output, of performance, the interview data provides us with a window into the student's thought processes. In related research, we have found this technique to be a source of genuine insights (Clement, Lochhead, Soloway [1980]).

Problem 1. Write a program which reads 10 integers and then prints out the average. Remember, the average of a series of numbers is the sum of those numbers divided by how many numbers there are in the series.

Problem 2. Write a program which repeatedly reads in integers until their sum is greater than 100. After reaching 100, the program should print out the average of the integers entered.

Problem 3. Write a program which repeatedly reads in integers until it reads the integer 99999. After seeing 99999, it should print out the correct average. That is, is should not count the final 99999.

Table 1. Problems used in our test instrument. These problems were given to an introductory programming class on the last day of the course. They are designed to test student knowledge of key differences between different loop constructs in Pascal.

Percentage of Students Which:	Problem 1 The for loop problem	Problem 2 The repeat loop problem	Problem 3 The while loop problem
Used for loop	16%	—	—
Used repeat loop	35.5%	53.6%	45.8%
Used while loop	35.5%	39.3%	41.7%
Used other constructions	12.9%	7.2%	12.6%
Answered problem correctly AND chose appropriate loop construct	9.7%	25%	20.8%
Answered problem correctly	36%	43%	38%

Table 2

The problems referred to in this table are listed in Table I. The above data is based on a sample size of 31 students.

```
program Student6_Problem3;

   var Count, Sum, Number : integer; Average : real;

   begin
   Count := 0;
   Sum := 0;
   Read (Number);
   while Number <> 99999 do
           begin
           Sum := Sum + Number;
           Count := Count + 1;
           Read (Number)
           end;
   Average := Sum / Count;
   Writeln (Average)
   end.
```

Figure 2. A stylistically correct solution to problem 3 in table 1. Note the need for two Read calls and the curious "process the last value, read the next value" semantics of the loop body. This program was minimally edited for presentation here. Students wrote these programs in a classroom. They were never submitted to a translator.

III.2 Performance Analysis: Use of Appropriate Loop Construct

In this course students were taught and used all 3 Pascal loop constructs: while, repeat, and for. Each of these constructs is appropriate in different situations (e.g., see Figure 1). Do novice programmers, in fact, distinguish between them and use them in the appropriate context? Answer, based on our data: no.

Table 2 lists the percentage of students using each particular loop construct on each problem; also listed are the percentage of students who used the appropriate loop construct and got the problem correct. The data on problem 1 are surprising. This problem is clearly a for loop problem, since the variable being tested is the counter variable. However, only 16% used a for loop; 35.5% used a repeat loop and 35.5% used a while loop, even though these constructs required the students to do more work by having to make explicit those operations done implicitly by the for loop (initialize counter, test counter for stopping, increment counter). Moreover, of the 36% who got the problem 1 correct, only 27% used the for loop. Clearly, choosing the appropriate loop construct did not contribute to correctly solving the problem.

Problem 2 (see Table 1) was a repeat loop problem; the variable that controlled the loop, "sum," needed to be assigned a value in the loop before it would be reasonable to test it. While more students did in fact choose the most appropriate loop construct, the difference between those choosing the repeat and those choosing the while was not statistically significant. Moreover, as in the above case, choosing the appropriate loop construct did not correlate with solving the problem correctly.

For problem 3 (Table 1), the appropriate loop construct is while; the loop must not be executed if the controlling variable has a specified value and therefore the test must be placed at the head of the loop. (This, in turn, requires a curious coding structure which we examine in the next section.) Here again, we see that the difference between those choosing a repeat loop and those choosing a while loop was not statistically significant. Nor again was the choice of the appropriate loop construct a predictor of the correctness of the solution.

Based on this simple test, it appears that novices do not distinguish between the Pascal loop structures as an expert might. A quote from a student we interviewed is appropriate here. When asked why he chose to use the while construct rather than one of the other two, he responded: "When I don't know what is going on, I use a while loop."

At first, we found the results on the for loop problem (Table 2, problem 1) counter-intuitive. After all, since the for loop does so much work automatically, we thought it would be the easiest to understand and use. On second thought, however, we decided that the automatic, implicit aspects of the for loop might be the sticky point. That is, since the for loop does do a number of actions automatically, students might be uncertain about all the details of the for loop's actions. Thus, in order to have some control over the beasty, students might choose a repeat or while loop and do the extra work to add the required looping machinery themselves. [1]

Quite simply, the difference between the repeat and while loops is subtle. Moreover, since with the appropriate extra machinery one construct can simulate the other, the distinction is hard to enforce. However, we feel that textbooks significantly contribute to the confusion. For example:

> The principle difference is this: in the WHILE statement, the loop condition is tested before each iteration of the loop; in the REPEAT statement, the loop condition is tested after each iteration of the loop.
>
> Findlay and Watt [1978]

---

[1] Some support for this interpretation comes from the following fact: students in this class were not taught the goto, and never constructed loops out of the basic constructs. Thus, these students may be unsure of the ingredients of a loop and might not be comfortable with all the magic implicit in a for loop.

> If the number of repetitions is known beforehand, i.e., before the repetitions are started, the for statement is the appropriate construct to express this situation; otherwise the while or repeat should be used. ... The statement [in a while body] is repeatedly executed until the expression becomes false. If its value is false at the beginning, the statement is not executed at all. ... The sequence of statements between the symbols repeat and until is repeatedly executed (at least once) until the expression becomes true.
>
> Jensen and Wirth [1974]

We feel that this is a syntax level description. What should be emphasized is the deep structure of this construction:

> Use a repeat loop if the controlling variable requires that it be assigned a value in the loop before it can reasonably be tested.

The frames and slots in Figure 1 reflect the close connection between the test in the loop and the loop construct, and the connection between the test and the problem.

A final comment; the reader might feel that the distinction between the three loop types is not important for a novice. To some extent, we agree. We wonder then why textbooks teach all 3.

III.3 Performance Analysis: Read i/Process i vs. Process i/Read Next-i

Let us now take a closer look at problem 3, the while loop problem. The stylistically correct solution requires a curious coding structure:

```
read first-value
while (test ith value)
     process ith value
     read next-ith value
```

The loop must not be executed if the test variable has the specified value, and this value could turn up on the first read; thus, a read outside the loop is necessary in order to get the loop started. However, this results in the loop processing being behind the read; it processes the ith input and then fetches the next-i. We call this structure "process i/read next-i."

Intuitively we felt this coding strategy to be unnecessarily awkward and downright confusing. A more "natural" coding strategy would be to read the ith value and then process it; we call this the "read i/process i" coding strategy. Do novice programmers use the stylistically correct coding strategy (process i/read next-i), or do they add extra machinery to a while or repeat loop (e.g., an embedded if test tied to a boolean variable) in order to force the code into a read i/process i structure?

Table 3 lists the performance of those students who attempted the problem with either a while or repeat loop. Of the nine who solved it correctly, only 2 used the stylistically correct process i/read next-i coding strategy. (See Figure 2 for a solution using this coding strategy.) To correctly solve the problem using either a repeat or while loop and the read i/process i coding strategy requires extra machinery; Figure 3 shows student programs which use this strategy. Nonetheless, the vast majority of students attempted this solution; given the extra complexity needed for a correct solution, it is not surprising that many failed.

It is tempting to conclude that with respect to these types of problems, Pascal requires that students circumvent their natural problem solving intuitions. Before we can actually assert this conclusion, more research needs to be done [1]. But, since we must live with Pascal for some time to come, it would only be responsible for teachers to explicitly teach their students about this peculiar coding strategy. Again, since humans are adaptive, we probably could learn to deal with this awkward and confusing construction.

III.4 Performance Analysis: Getting a New Value

In all 3 problems (Table 1), a correct solution required that the program get a new value with a read. 23% of all the student written programs did not perform this function correctly. Often students try to get the previous or next

	Read i/Process i used			Process i/Read Next-i used		Other
	repeat loop	while loop	other	repeat loop	while loop	
Correct	4	2			2	1
Incorrect	3	5	4			2

Table 3

The numbers in this table refer to the actual number of students, not percentages.

```
program Student7_Problem3;

   var N, Sum, X : integer;
       Average : real;
       Stop : boolean;

   begin
   Stop := false;
   N := 0;
   Sum := 0;
   while not Stop do
           begin
           Read (X);
           if X = 99999
                   then Stop := true
                   else begin
                        Sum := Sum + X;
                        N := N + 1
                        end
           end;
   Average := Sum / N;
   Writeln (Average)
   end.
```

```
program Student16_Problem3;

   var Count, Sum, Num : integer; Average : real;

   begin
   Count := -1;
   Sum := 0;
   repeat
           Count := Count + 1;
           Read (Num);
           Sum := Sum + Num
   until Num = 99999;
   Sum := Sum - 99999;
   Average := Sum / Count
   end.
```

Figure 3. These programs are attempts at problem 3 described in table 1. They are typical of the contortions students will go through to make this problem fall into a "read a value, process that value" Frame. These programs have been minimally edited for presentation here. Students wrote these programs in a classroom. They were never submitted to a translator.

value of a variable by subtracting or adding one (see Figure 4). [2] We also found programs in which we felt students assumed that each *use* of Next_value automatically retrieved a new value.

As we indicated in Figure 1, getting a new value is different than, say, accumulating a total. Thus, perhaps students committing the above errors did not understand that *read* is actually just a special case of assignment. If so, then a language which treated I/O calls as special values which can be assigned to or from might be more palatable to beginning programmers, e.g.,

```
  New_value := Read_from_terminal, or,
  Write_to_terminal   :=   Running_sum
Count.
```

Another possible mind bug which could result in some of the observed errors would be that students incorrectly overgeneralized from the Counter Variable Frame. That is, since the next value of a variable functioning as a counter can be retrieved by simply adding a 1 to the variable, why not get the next value of *any* variable by simply adding a 1 to it! While reasonable, this is incorrect. This type of overgeneralization could be predicted from the relationship of the frames in Figure 1.

III.5 Performance Analysis: The Different Role of the Assignment Statement in the Counter and the Running Total Templates

If one chose to use either a *repeat* loop or a *while* loop in any one of the three problems, one would need to explicitly keep track of at least two quantities: (1) the number of numbers which were read in, and (2) a tally of the sum of the numbers read in. In both cases, one would use a particular type of assignment statement which facilitated a running sum, e.g., Running_total := Running_total + New_value. In the former case, New_value would be the constant 1, while in the latter case New_value would be dependent on the value read in.

Since the underlying programming language construct is the same in both cases, one might think that if a student used the construct correctly in the counter case, then the student would understand the construct and would most likely be able to use it correctly in the analogous case of tallying up the sum. However, the performance results in Table 4 portray a different picture. Namely, significantly more students constructed a correct assignment statement for the counter action than could do so for the running total action.

Why? The knowledge network in Figure 1 suggests, in fact, that the Counter Template and the Running_Total Template are distinct, since the functions they perform, while similar, are still different. Moreover, the Running_Total is more complicated since the New_Value Template needs to be integrated to provide a correct solution. Thus, one possible explanation of the performance difference would be that the students did not fully understand how these two functions were integrated; this added complexity was responsible, then, for the poorer performance.

Yet another interpretation consistent with the frame organization suggested by Figure 1 is the following: students understand the counter action as a whole, and do not decompose I := I + 1 into a left hand variable having its value change by the right hand expression.

---

[1] We have designed and pilot-tested the following experiment: first, we ask all students to write a plan or design for problem 3 in Table 1 (the same one examined in this section), in a language other than a programming language. We then ask half the students to write the program in Pascal. For the other half of the group, we provide a one page description of the Ada *loop* ... *exit* loop construct. While the sample size was small (13 students), the data are suggestive: invariably the *plan* of the students was worded in terms of a read i/process i. However, the Pascal versions were typically coded with a process i/read next-i strategy. But, those programs written using the Ada *loop* ... *exit* were coded using the read i/process i strategy. Thus, the program coded in Ada more closely matched the students' plans than did those program coded in Pascal. We plan to run this experiment on a larger group.

[2] "Backing up" may be needed when a student does the *while* loop problem (problem 3) with a *repeat* loop.

	Sample Size	Percentage		Statistical Significance
		Correct Running-Total Assignment	Correct Counter Update Assignment	
Overall (across all problems)	69	68%	81%	.019*
Problem 1	22	59%	77%	.042*
Problem 2	26	69%	69%	1.00
Problem 3	21	76%	100%	.021*

Table 4

The asterisks indicate statistically significant differences.

```
program Student19_Problem1;

   var Num, Prev_num, Count : integer;

   begin
   Count := 0;
   Read (Num);
   Sum := 0;
   repeat
         Prev_num := Num - 1;
         Sum := Num + Prev_num;
         Sum := Sum + 1;
         Count := Count + 1;
   until Count = 10;
   Average := Sum / Count;
   Writeln ('Average of ten integers is equal to ':2)
   end.
```

```
program Student30_Problem2;

   var N, Sum, Score : integer;  Mean : real;

   begin
   N := 0;
   Sum := 0;
   Score := 0;
   while (Sum <= 100) do
         begin
         Score := Score + 1;
         Sum := Sum + Score;
         N := N + 1
         end;
   Mean := Sum / N;
   Writeln ('the mean = ', Mean:10:10)
   end.
```

Figure 4. These programs are attempts at the problems described in table 1. They illustrate student problems with getting a New_value. These programs have been minimally edited for presentation here. Students wrote these programs in a classroom. They were never submitted to a translator.

That is, students do not view I := I + 1 as an example of an assignment statement.[1] Thus, when faced with developing an assignment statement for the running total function students must really confront their understanding of the particular type of assignment statement needed in this context; the poorer performance in this situation reflects a misunderstanding of how the assignment statement works.

III.6 Performance Analysis: The "Demon" in the while loop test

Based on our examination of student programs, and on an analysis of the individual interview, we felt that there was a great deal of confusion surrounding the time at which the terminating test in the while loop gets evaluated: is it evaluated once at the top of the loop, or is the test continually evaluated during the execution of the body of the loop? The program given below was also on a written test taken by the 31 summer school students.

```
program Problem4;
   var Count : integer;
   begin
   Count := 0;
   while Count < 7 do
            begin
            Writeln ('*');
            Count := Count + 1;
            Writeln ('/')
            end
   end.
```

If the students felt that the terminating test was evaluated continually, then the loop should terminate before an '/' were printed, thus providing one more '*' and '/'.[2] In otherwords, it is as if the test were a demon watching the statements in the loop body, and waiting for its condition to become true. Of the 31 students, 34% made the above mistake. Since while is commonly used in programs and in the instruction, and since it was the end of the semester, we felt that this was a surprisingly high percentage.

The basis for this confusion is grounded in the mismatch between the semantics of while in a programming language context, and the semantics -- the meaning -- of 'while' in every day experience. In the latter case, 'while' has a global sense: during the course of some event. In contrast, the programming language while requires a local, narrow interpretation: at a specific point in time. Clearly, the names of programming language constructs must rely on real world semantics of their analogs. However, care ought to be exercised in their selection. Since the likelihood of renaming the while construct in Pascal is small, educators must take note of this error, and pay attention to it in their instruction.

IV. Concluding Remarks

In this paper we first argued for the need of empirical research coupled with crisp, detailed theories of the programming process. We then went on to develop an explicit knowledge network which represents what programming experts might know about looping and assignment. We argued that since experts apparently had knowledge, such as plan types, which were structured into bundles, it was only responsible that we as educators teach our students this type of knowledge. Based on this knowledge network, we analyzed buggy programs collected from introductory Pascal students. We expressed the analysis in terms of mind bugs -- misconceptions -- that resulted in program bugs. In addition, we found that Pascal construction themselves might be the cause of some program bugs, since they trigger inappropriate and misleading conceptions in a student's mind.

Without a doubt, the claims we have made are controversial -- in fact, they may even be incorrect. However, we strongly feel that dialogue must be encouraged on this type of research, if computer science education, programming language design, and computer literacy, are to be advanced.

---

[1] Problem 3 suggest some intriguing corroborating evidence. More students got the count action correct (e.g., I := I + 1) than got the count initialization correct (e.g., I := 1)! Maybe this was due to sloppiness. However, if I := I + 1 is a unit unto itself, then possibly the students do not see the need to initialize the variable.

[2] We were not interested in the actual number of '*' and '/', because we were not studying the off-by-one bug in this particular problem.

V. Acknowledgements

We would like to express our appreciation to Steven Levitan for his helpful comments and David Lee for his diligent support work.

V. Bibliography

Brown, J.S. and Burton, R.R. (1978) "Diagnostic Models for Procedural Bugs in Mathematics," Cognitive Science, June.

Clement, J., Lochhead, J. and Soloway, E. (1980) "Positive Effects of Computer Programming on Students' Understanding of Variables and Equations," Proc. of National ACM Conference, Nashville.

Collins, A. (1978) "Explicating the Tacit Knowledge in Teaching and Learning," presented at the American Education Research Association (also BBN Technical Report 3889).

Dijkstra, E.W. (1976) A Discipline of Programming, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Dunsmore, H.E. and Gannon, J.D. (1978) "Programming Factors - Language Features that Help Explain Programming Complexity," Proc. of National ACM Conference.

Findlay, William and Watt, David A. (1978) Pascal: An Introduction to Methodical Programming, Computer Science Press, Inc., Potomac, Maryland.

Friedman, Daniel (1974) The Little LISPer, Science Research Associates, Menlo Park, Calif.

Gannon, J.D. (1978) "Characteristic Errors in Programming Languages," Proc. of 1978 Annual Conference of the ACM, Washington, D.C.

Goldstein, I. (1974) "Understanding Simple Picture Programs," Technical Report AI-TR-294, M.I.T. A.I. Lab, Cambridge.

Jensen, Kathleen and Wirth, Niklaus (1974) Pascal User Manual and Report, Springer-Verlag, New York.

Kurtz, T.F. (1978) "BASIC," appeared in the Proceedings of the ACM SIGPLAN History of Programming Languages Conference, SIGPLAN Notices vol. 13, num 8, Aug.

Larkin, J., McDermott, J., Simon, D. and Simon, H. (1980) "Expert and Novice Performance in Solving Physics Problems," Science, 208.

Ledgard, H., Whiteside, J., Singer, A. and Seymour, W. (1981) "Report on an Experiment on the Design of Interactive Command Languages," to appear in Communications of the ACM.

Mayer, R. (1980) "Contributions of Cognitive Science and Related Research in Learning to the Design of Computer Literacy Curricula," Conference on National Computer Literacy Goals for 1985, Virginia.

McCracken, Dan (1976) forward to APL: An Interactive Approach, by Leonard Gilman and Allen J. Rose, John Wiley and Sons, New York.

Miller, Lance (1978) "Behavioral Studies of the Programming Process," IBM Technical Report RC7367, Yorktown Heights, New York.

Miller, Mark L. (1978) "A Structured Planning and Debugging Environment for Elementary Programming," Int. J. Man-Machine Studies, 11, pp. 79-95.

Minsky, M. (1975) "A Framework for Representing Knowledge," in The Psychology of Computer Vision (P.H. Winston, ed.), McGraw-Hill, New York.

Nilsson, N. (1980) Principles of Artificial Intelligence, Tioga Publishing Company, Palo Alto, California.

Papert, S. (1980) Mindstorms, Children, Computers and Powerful Ideas, Basic Books, Inc., New York.

Polya, G. (1973) How To Solve It, 2nd Ed., Princeton University Press, New Jersey.

Rissland, (Michener) E. (1978) "Understanding Understanding Mathematics," Cognitive Science, vol. 2, no. 4.

Rissland, E.R. and Soloway, E.M. (1980) "Generating Examples in LISP: Data and Programs," Technical Report 80-07, Dept. of Computer and Information Science, Univ. of Mass., Amherst.

Shneiderman, B. (1980) Software Psychology, Human Factors in Computer and Information Systems, Winthrop Publishers, Inc., Cambridge.

Soloway, E. and Woolf, B. (1980) "Problems, Plans, and Programs," in Proc. of Eleventh ACM Technical Symposium on Computer Science Education, Kansas City.

Teitelbaum, T. and Reps, T. (1980) "The Cornell Program Synthesizer: A Syntax-Directed Programming Environment," Department of Computer Science, Cornell University, Technical Report 80-421, May.

Waters, R.C. (1979) "A Method for Analyzing Loop Programs," IEEE Trans. on Software Engineering, SE0:3, May.

Welty, C. and Stemple, D.W. (1981) "Human Factors Comparison of a Procedural and a Nonprocedural Query Language," to appear in Trans. on Database Systems.

Winston, P. and Horn, B.K.P. (1981) LISP, Addison-Wesley Publishing Co., Reading, Massachusetts.

Wirth, N. (1976) Algorithms + Data Structures = Programs, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Wirth, N. (1977) "The Programming Language Pascal," Acta Informatica, 1, pp. 35-63.

SCIENCE LITERACY IN THE PUBLIC LIBRARY - BATTERIES AND BULBS

Arnold Arons
Department of Physics
University of Washington
Seattle, Washington 98195

Alfred Bork
(714) 833-6911
Stephen Franklin
Barry L. Kurtz
Educational Technology Center
University of California
Irvine, California 92717

Francis Collea
Department of Sccience Education
California State University
Fullerton, California 92634

The Educational Technology Center at the University of California, Irvine, is currently conducting a project in science literacy for public environments by focusing on the public library. These computer-based modules run on personal computers, intended for a wide variety of environments and a wide audience. These modules attempt to tackle deep problems with regard to what science is all about, what a scientific theory is, how theories are developed, how theories are verified, how theories are modified, and the role of measurement. Although public environments are the initial focus, we believe that the materials will be very useful in school programs at a wide variety of levels and eventually can be distributed for the home personal computer market.

This paper reports in detail on one of the sets of modules developed for this project.

Batteries and Bulbs

A number of programs at levels varying from primary school through university have used a simple physical situation, the lighting of a flashlight bulb using batteries and wires, to introduce to students the notion of a model. The Elementary Science Study (ESS) materials are an example. It has also been used, as indicated, in other programs.

This material is based on guided discovery. The student is not *told* the necessary information, as in typical lectures or textbooks. Rather, with a good teacher in the classroom, the student works by discovering, while guided and moderated by the teacher. One doesn't simply turn the student loose with the apparatus and expect major laws of science to be developed (this is clearly impractical). The teacher acts as a guide, a facilitator, and a helper.

The battery and bulbs material does not intend to teach people simple information about batteries and bulbs or information about the behavior of electrical circuits. The aim is *not* to introduce the important terminology in the area, although such vocabulary does eventually get introduced. Rather, the module tries to develop the notion of a *scientific model* as the way of dealing with a range of phenomena. When used correctly, the existing material emphasizes the role of experimentation, the necessity of forming hypotheses and building models, the central role of using models to make predictions, and the attempts to verify predictions through experimentation, modifying the models or constructing new ones.

The difficulty with discovery or inquiry-mode materials of this kind is that they often require a teacher who understands the processes of science and is willing (and able) to let each student progress at individual rates towards these ideas. The sad fact is, however, that not all teachers can proceed in such a fashion.

Implementing many of the new and excellent science curricula at levels from kindergarten through the university is difficult for traditional teachers. Changing the behavior of an entire group of science teachers is extremely difficult, and teacher training has been one of the major problems with the new curriculum projects. These new materials, often excellent, do not reach their objectives in many classrooms. Although these objectives are clearly explained in teacher materials, many teachers simply do not sufficiently understand the processes of science to use the materials effectively. In addition, to devote enough time to _each_ student is often difficult or impossible. Further, teachers are reluctant to follow an approach that appears to mean additional work on their part.

Furthermore, if many people who are already through the school system are to be brought to a better understanding of the nature of science, they must have other educational strategies available to them. The problems of adult education are very much on our minds in the present project. We believe it wise to build on existing experiences. We chose batteries and bulbs because there has been so much thought put into the development of these materials. But we believe that we can bring these materials to a much wider audience, and we can successfully solve the problem of teachers who are not trained for the necessary approaches.

Fundamental Ideas Developed

The materials do _not_ have as their primary purpose the development of terminology or concepts. Nevertheless the notion of _how_ concepts are developed within science is an important issue in scientific literacy, and so is given attention.

In the batteries and bulbs module, current and resistance are major concepts. In following recent strategies at the Educational Technology Center, we do not introduce these terms immediately. We avoid the use of vocabulary, contrary to much of the teaching of science in American schools, until a sound and demonstrated _need_ for the terminology has been established in the student's mind. That is, the concept should not appear until the notions underlying it have already been developed and shown to lead to useful consequences.

In introducing the concepts of current and resistance, we therefore proceed very slowly with the student, interacting in an experience gathering environment. These concepts themselves are introduced only in a qualitative situation and are never assigned any numerical value within this program. Thus, such issues as units never play a role. The currents are compared; students are brought to the point where they can empirically say that the current in one situation is greater than the current in another situation. This comparison is based on the brightness of light bulbs. Four levels of brightness are sufficient--full, medium, low, and off.

Resistance is also developed in the same qualitative manner, without numbers. Students do learn that the resistance of two bulbs in series will be greater than the resistance of a single bulb. Furthermore, they see several different types of wires, such as copper wire and nichrome wire, and study them from the standpoint of resistance.

The model of an electrical circuit is gradually introduced, although again with little emphasis on terminology. We believe, as already indicated, that one of the major problems with learning about science is the extensive dependence on vocabulary.

"Real" versus "Simulated" Equipment

One of the interesting issues considered in developing this material is whether the student should have the actual equipment--batteries, bulbs, and wires--available or whether equipment should be simulated within the computer program. We have used a variety of tactics in different modules developed recently in the Educational Technology Center. One tactic is to use equipment along with computer material. Another module under development for junior high science employs a "Whirlybird" out of the Science Curriculum Improvement Studies material.

However, with batteries and bulbs the decision was that we would _not_ use the physical apparatus, although we would mention it within the program and suggest users experiment on their own. Several members of the development team were somewhat apprehensive about this choice. We see both advantages and disadvantages in our approach. Many science teachers consider it important to use the actual equipment to deal as closely as possible with the phenomena. On the other hand, for the purposes of this particular module, the somewhat idealized equipment that we supply within the program had some advantages. Thus, we did not need to deal with the problem that not all batteries are the same, not all bulbs are the same, etc. Occasionally when the actual equipment is used with students, one has to put in spurious discussions explaining away some of the things that happen. With the apparatus we must assure identical bulbs and equally charged batteries.

Although we are not initially conducting an experimental test on actual versus simulated equipment, we hope that the batteries and bulbs unit may allow further research into this problem of equipment in connection with scientific education. We intend to encourage experimental studies comparing computer materials without equipment, computer materials with equipment, and traditional instruction (no computer) with equipment. Many scientists have strong and emotional views on this issue. The research literature seems to have little in the way of careful studies.

Structure of the Sequence

Since this and the other materials in this particular project are designed for public environments in situations in which no assistance is available, we decided that the materials should be broken into a series of quite small modules. The basic notion of these small modules was that students may not have a long attention span for any one use, but that one could expect over a period of time return visits for some students who become interested in the material. Students can work through as much material as they are comfortable with in a single session. The modularity of the dialog is an important issue.

The modules are ten or fifteen minutes in length for the average student. However, as with any good computer-based learning material, we expect this time to vary considerably from student to student. The modules in the batteries and bulbs sequence are as follows:

Module 1 - Light the Bulb
Module 2 - Battery and Bulb Arrangements
Module 3 - Other Things in the Circuit
Module 4 - A Scientific Model of a Circuit
Module 5 - Two Bulbs
Module 6 - Obstructing Current
Module 7 - Current Paths
Module 8 - Multiple Paths
Module 9 - Playing with Circuits

The student will initially see the program in what is called the "attract mode." This mode, used in all the materials in this project, lures passers-by in the public library into sitting down and trying the program. Because of the general interest people have in computers, we believe that this will be relatively easy. The programs must appear nonthreatening, and they must not have any large number of operating details before getting into the materials. The attract modes in these programs are often made from interesting visual materials within the program.

When the student does sit down and follows the direction (on the screen in the attract mode) to press a key, a map of the modules available appears. The student uses a built-in pointing device to indicate where within the structure he or she would like to start. We would urge beginners to start at the beginning. But, since this is a free environment, there is no assurance that they will do so.

As already indicated, the materials must be entirely self-contained. We do not expect the librarians to spend time aiding students. Furthermore, if the material is used in other environments, there may not even be any knowledgeable people present. So the dialog must make certain that the students can use the equipment.

A number of strategies are being used. First, we make extensive use of timed "reads." That is, if the student has not replied to a question or a prompt after a given period of time, the program assumes control and gives the student a message. In early parts of the program, these may simply be encouraging input or telling the student what is expected, such as pressing Return when the input is complete. Our input software, developed in our NSF CAUSE project, distinguishes between someone who is not typing at all and someone who has typed something but has not pressed the return key. Furthermore, the student who is actively typing, even though there is a sizable time before typing began, will not be timed out, but will be allowed to continue typing. All of these are simple, human engineering strategies used in the Educational Technology Center at Irvine for some time.

There is also a mechanism of assuring that when a student leaves a display, the program will eventually return to the attract mode to await another user. Note that the machine is always on and that no program needs to be loaded by the user.

Production Process

These materials were produced by the process which has been used for all recent materials within the Educational Technology Center. One of the major interests in the Center has been to develop a reasonable production strategy, one that can be extended eventually to future needs for large-scale production. This process has been described in other papers associated with the Center.[2] The first stage was the pedagogical development stage, the full specification of the script which tells everything about how the program is to behave with the user. The principal pedagogical developers were Arnold Arons, Alfred Bork, and Barry Kurtz. Francis Collea and Stephen Franklin also contributed valuable advice and assistance, particularly in the early stages of the development. Development at this stage took approximately one week, with the authors developing the material collectively.

As with all of our activities, we concentrate at this stage on the instructional issues; programming issues are rarely considered. The group worked together, rather than fragmenting into several different groups. We argue that we produce much superior material with several people working together than any one person does individually.

The next stage involves the actual development of the running computer materials. With the batteries and bulbs modules, the principal programmers involved were Mark Gugan, Adam Beneschan, James Zarbock, and Miichael Potter. We work within the UCSD Pascal system, supplemented with additional software tools developed at the Educational Technology Center. The programmers may discover some difficulties or areas overlooked by the pedagogical developers.

The computer on which the development takes place is a Terak 8510/a. We are also using the Terak for the initial testing phase, but we plan to deliver the materials on a variety of small personal computers. We believe it increasingly important to distinguish between hardware needed for development and hardware needed for delivery. If the material is developed with reasonable strategies, moving it to new delivery machines does not produce any tremendous difficulties.

After the materials are running, much additional examination of them goes on by both the programmers and the authors. Other people at the Educational Technology Center look at these materials and offer advice. Arnold Arons, in a return visit, made a detailed study of the unit with Barry Kurtz and the suggested changes are now being implemented.

At this stage in the process all the testing has been informal internal testing. The next stage is that of moving to the target population. As of this writing, we are just starting that process.

The follow-up stages involve gathering information during this initial process of testing on the target audience and reworking the dialogs based on this information. Sometimes several cycles are necessary for this.

We can next proceed to larger testing. That step is not planned within the present project. Rather, we believe that it is better for materials to be used for several years and to have reached a stable form before summative evaluation. Our main aim is developing greater scientific literacy in a wide group of people in the general population. Hence, any final testing must look carefully into this particular issue.

The materials will be available for use during the National Educational Computing Conference, along with a wide range of other recently developed computer-based learning materials produced by the Educational Technology Center. The project welcomes visitors and is happy to send additional information about its activities. It also publishes a newsletter and welcomes requests to be put on the mailing for the newsletter.

References

1. Elementary Science Study - Batteries and Bulbs, Webster Division, McGraw-Hill Book Company.
   Arons, A., The Various Languages, Oxford, 1977, Chapter 9.
   ----- "Phenomology and Logical Reasoning in Introductory Physics Courses," American Journal of Physics (in press).

2. Bork, A. Educational Technology Center at the University of California,

Irvine, IFIPS, London, Summer 1979.
Bork, A. and Franklin, S., The Educational Technology Center, National Educational Computing Confference, 1980.

TEACHING SIMULATION TECHNIQUES WITH MICROCOMPUTERS

J. D. Spain
Department of Biological Sciences
Michigan Technological University
Houghton, Michigan 49931
(906) 487-2029

ABSTRACT

A course called Biological Simulation Techniques has been offered at Michigan Tech for about 10 years. Initially, Olivetti programmable calculators were used, but since these use a machine-dependent language, there was little carry-over value for students. For this reason the course was changed to use Basic language, and this change has been a key factor in developments described in this paper. Subject matter now includes most major modeling techniques and provides examples drawn from all the major fields of biology. Using Basic naturally led to the introduction of microcomputers. A text has evolved along with the course and is now in final stages of publication. The course has been the subject of several workshops held at Michigan Tech and now serves as a model for similar courses being developed at other colleges and universities. The potential role of computer modeling and simulation in undergraduate life-science curricula will continue to expand as biology continues to become more quantitatively oriented.

BACKGROUND

The Department of Biological Sciences at Michigan Tech began a formal course in simulation techniques in 1972. From the start, students learned best by personal involvement in model development. To encourage this activity, 80 percent of the final grade was based on exercises submitted by the students. Olivetti programmable calculators were initially used for programming exercises. These early experiences with the course were reported to the Conference on Computers in the Undergraduate Curricula held at East Lansing in June 1977. Because the Olivetti programming was done in a machine specific language, we decided students would benefit more by using the university's time-sharing system and programming language, a key factor in subsequent course development. Freed from specific hardware requirements, we could develop materials with general application in undergraduate life-science curricula.

BASIC AS A SIMULATION LANGUAGE

Basic proved to be a very satisfactory language for instructional simulations.

1) It is relatively simple. Typically students write programs within the first week and expand their capabilities, largely on their own, throughout the course. The instructor can emphasize modeling and simulation techniques rather than programming. We expect that as more students are exposed to Basic in high school we will be able to develop highly sophisticated models during the short period of one quarter.

2) Unlike languages specifically designed for simulation, Basic can be used to explore a complete range of modeling techniques rather than emphasizing one at the expense of others. We assume that students, after completing a survey course such as this, would move easily to specific simulation languages as the need arises.

3) Because Basic is an interactive language, students can develop a model and then easily explore the effects of changing parameters. This feature also greatly facilitates the modification or "fine tuning" of models. In many ways, Basic is nearly ideal for computer modeling.

4) Finally, Basic is the language most generally used on microcomputers. This made it very convenient to adapt the course to use microcomputers when they

became available at a cost competitive with time-sharing terminals.

Thus, the transition to Basic made possible the development of a simulation course in a form that is transportable to other colleges.

TRANSITION TO MICROCOMPUTERS

For three years students used the UNIVAC 1110, primarily in the time-share mode. This approach was not totally successful for a variety of reasons. Students without previous computer experience often dropped the course as a result of the initial trauma of interacting with the computer. This problem was often compounded by a communication breakdown between the computer center, the instructor, and the student. In any case, use of UNIVAC involved difficulties not previously encountered when the use of programmable calculators was controlled by the instructor. To avoid these difficulties, we looked for alternatives which offered instructor control. Other problems, including crowding in the computer center and the long distance between students and the instructor when questions arose, would also be solved. We considered purchase of additional time-sharing terminals to improve the situation. However, we found that microcomputers capable of graphics display, hard-copy output, and independent operation could be obtained for little more than a time-share terminal - the choice was obvious. Support for the purchase of additional equipment was obtained from the National Science Foundation, Instructional Scientific Equipment Program under grant No. SER-8013406.

HARDWARE REQUIREMENTS

The Apple II microcomputer was selected because it provides a relatively high resolution graphic display which is essential as an end-product for most modeling exercises. Each student station is provided with the microcomputer with Applesoft ROM, 48K bytes of memory, a disk drive, a 9" monitor, and silentype printer. Five stations are provided for a class of 40 students. Each student is scheduled for 3 hours of computer time each week. The lab is open about 50 hours during the week so students have adequate time to carry out the assigned work.

Equipment problems have been no more numerous than would be expected from new equipment. The modular nature of the Apple system has made it easy to pin-point problems and to replace defective components.

PROGRAMMING AIDS

At the beginning of the course, each student submits a floppy disk to the instructor for initialization with several programming aids. These aids include:

1) RENUMBER - an Apple program for resequencing and merging Applesoft programs.

2) GRAPH - a combination of subroutines used to draw and label axes, plot lines, or points, and to write character strings on high resolution graphics.

3) CURVEFIT - a program that permits the user to fit 9 different model equations to a set of x,y data by least squares, select the best-fit by statistics, and obtain hard-copy graphic output.

4) POLYFIT - a program that fits five polynomial equations to a set of x,y data, selects the best-fit, and provides hard-copy of statistics and a graph of the best-fitting model in relation to the data.

5) POISSON - a program that compares computer generated data with those expected from the Poisson distribution.

TEXT

One problem in teaching the course is a lack of appropriate texts. Although several deal with biological modeling, essentially none are designed to be used by students learning the practical aspects of converting mathematical models into computer simulations, an important step students only learn by doing. Thus, we began to develop and to test a large number of modeling exercises illustrating all the major modeling approaches applicable to biological systems. This material has been collected and will be published shortly as a book by Addison-Wesley.

The course is divided into three parts. The first deals with simple equation models based on an analytical solution of rate or steady-state equations. The second part discussed multicomponent deterministic models. They are generally based on a numerical solution of differential equations, although several other numerical techniques are included. The third part emphasizes discrete event models which employ the Monte Carlo technique to make decisions based on random numbers. In each of the three parts, exercises and examples are drawn from all fields of biology, from the molecular to the population level. Emphasis is placed on the integrative nature of mathematics as it relates to biology.

FACULTY WORKSHOPS

For the past three years Michigan Tech has carried on college faculty workshops dealing with biological simulation techniques and the general application of microcomputers to college biology instruction. Approximately 50 individuals representing over 40 colleges and universities have participated. The primary objective of the 1981 workshop is to review a series of instructional simulations which have been developed at Michigan Tech with support from the National Science Foundation, Development in Science Education Program (SED-7919051).

POTENTIAL FOR COMPUTER SIMULATION IN UNDERGRADUATE LIFE SCIENCE CURRICULA

Undergraduate biology instruction is becoming much more quantitative. Many bioscience programs have increased requirements for mathematics, statistics, and computer science. In addition, many schools now require two or more years of chemistry and physics for the B.S. degree in biology. In many fields and at many universities, graduate study in life-science has become highly mathematical and modeling oriented. Examination of life systems strongly suggests that the potential for mathematical applications in biology may ultimately far exceed all the other sciences. Clearly, we should begin providing a stronger mathematical basis for most biology coursework. The addition of a computer simulation course provides at least one measure of meeting this important need. A simulation course provides information and concepts that are complementary to the topics which are typically taught in biometrics courses. At Michigan Tech, both biometrics and computer simulation are taught on the senior level, and quantitatively oriented biology students usually take both. As biology becomes more mathematically oriented, it is only natural that courses such as biometrics and simulation will play increasingly important roles in bioscience curricula.

ACKNOWLEDGEMENT

Thanks go to Dr. James Horton, Department of Biology, California State College at Bakersfield for critical review of this paper.

COMPUTER-ASSISTED INSTRUCTION IN SECONDARY PHYSICAL SCIENCE: AN EVALUATION OF STUDENT ACHIEVEMENT AND ATTITUDES

Gary H. Marks
Science Education Center, EDB 340
University of Texas at Austin
Austin, Texas 78712

Rolland Bartholomew
Science Education Center, EDB 340
University of Texas at Austin
Austin, Texas 78712

The use of computers in education has slowly increased over the past twenty years. A number of recent changes in technology and education has now greatly increased the educator's interest in computers for learning and teaching:

1) The low cost of microcomputers merits rethinking the uses of technology in education.
2) Inexpensive communications may shift the role of centralized educational computing.
3) Incentives will improve for commercial production of materials for training and continuing education.
4) Home computing, combined with other technology, will support a trend toward more education in the home (Zinn, 1978).

These changes coupled with such predictions that "within the next decade every secondary school in the country will have access to a computer system for some type of administrative and/or instructional application" (Bukowski and Korotkin, 1976) make it clear that computer-assisted instruction (CAI) will play an important role in the future of education. Yet individual and institutional acceptance of CAI has not been overwhelming. Leaders in education recognize that adoption of a new innovation, such as CAI, should be preceded by studies on the impact of this new technology.

With these developments in CAI and the related problems in mind, researchers began to consider the potential value CAI may hold for secondary physical science students. Can secondary students learn with CAI? What would be the students' reactions toward CAI?

The educational promise of CAI lies in its ability to individualize and personalize the instructional process. CAI lessons can test as well as tutor by encouraging students to become active participants in their own learning. Students work at their own pace while the computer monitors their progress. Students are kept informed of their progress through immediate response and achievement summaries.

These attributes seem beneficial to all content areas, including physical science. After making a search of existing physical science courses, commercially available CAI materials and related literature, we discovered a void exists in the use of CAI in secondary physical science. Can this void be filled? If so, what reactions might secondary physical science students have toward CAI materials?

Purpose of the Study

This study was designed to: (1) develop a CAI lesson that involves the study of a common topic of physical science for use in secondary science classes; (2) test the materials in a natural setting; (3) determine the effectiveness of the CAI lesson based on achievement tests related to specific lesson objectives; and (4) identify and measure the reaction of students to a CAI lesson.

Can CAI be used to supplement the teaching of secondary physical science? To answer this question, a CAI lesson about introductory electricity was developed and tested for a ninth grade physical science course. In testing the material, data were gathered to answer these questions:

1) Were the content objectives for each part of the lesson fulfilled?
2) Do students' pre- and post-test achievement scores indicate they learned the CAI lesson material?
3) What were the students' reactions toward the CAI lesson?
4) Was there a relationship between student achievement scores and student reactions towards the CAI lesson?

Description of the Study

To conduct this study it was necessary to:
1) select a lesson topic, 2) develop and program the lesson, 3) write a study guide, and 4) develop and administer the evaluation instruments. The CAI lesson materials consisted of a four-part lesson plan; a TRS-80 Level I microcomputer; a study guide; wires, switches, batteries, bulbs, and circuit boards; meters: voltmeter and ohmmeter; and a calculator.

The lesson topic had to be adaptable to the tutorial and drill/practice modes of CAI, and use a variety of manipulative instructional aids. Electric circuitry satisfies all these objectives, and was selected as the lesson topic.

The lesson was developed in four major parts. The first part was a short tutorial, interactive program designed to familiarize the student with how to operate the microcomputer. The student was introduced to the computer keys and shown how to respond to computer questions. This part of the lesson was designed to relieve the student of any initial anxiety about the use of the computer.

The second part introduced the student to electrical symbols, circuit diagrams, and elementary terminology. This was done in conjunction with the student study guide. A progress check was made at the end of this part to determine if the objectives had been attained. If not, the student repeated this part.

The third part explained Ohm's Law and allowed the student to practice problems of this kind for a series circuit. The computer randomly generated Ohm's Law problems for each student. A hand calculator was allowed since this lesson was not designed to improve mathematical abilities. If a minimal score of 75% was not achieved, the concept of Ohm's Law was reviewed and the student completed another progress check.

In the fourth part, students built series circuits according to diagrams presented in the study guide. After getting the light bulbs to light, meter readings were taken in different parts of the circuit. These readings were recorded in the study guide and their accuracy verified with the computer. If the readings were not within an acceptable range, the computer program gave the correct answer and explained why this answer was correct.

The lesson was programmed on a Radio Shack TRS-80 Level I Basic microcomputer. Throughout the lesson a study guide was used in conjunction with the computer to provide more detailed instructions and to record information.

The objectives we identified represented a variety of cognitive levels as described by Bloom (1956), and served both as the basis for development of the evaluation instruments and the lesson content material. Each test item was related to a specific objective, and the same items were used in the pre- and the post-test.

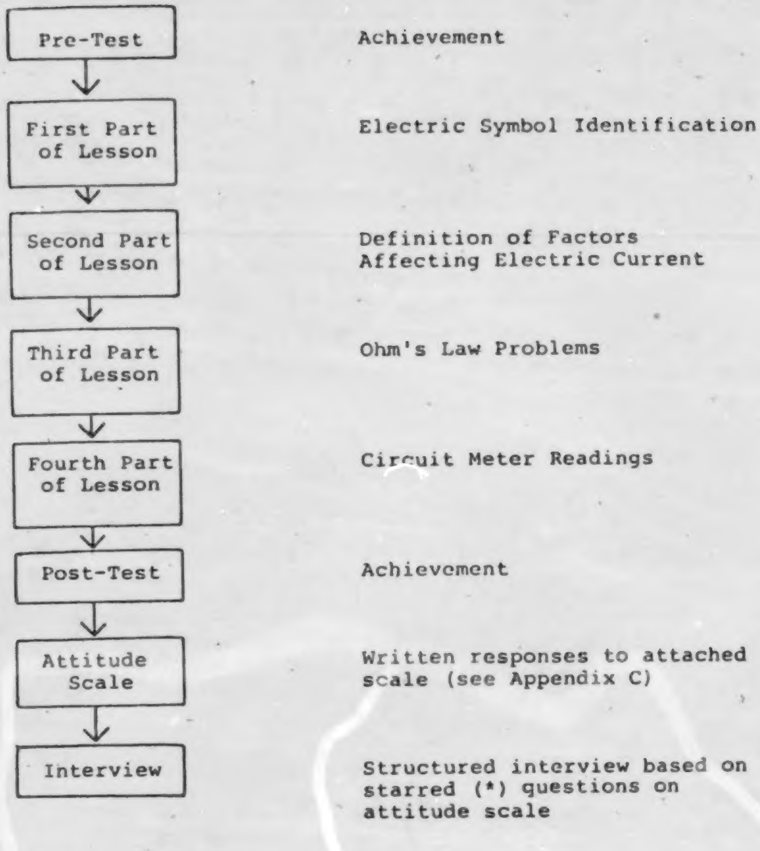

Pre-Test
Achievement
First Part of Lesson
Electric Symbol Identification
Second Part of Lesson
Definition of Factors Affecting Electric Current
Third Part of Lesson
Ohm's Law Problems
Fourth Part of Lesson
Circuit Meter Readings
Post-Test
Achievement
Attitude Scale
Written responses to attached scale (see Appendix C)
Interview
Structured interview based on starred (*) questions on attitude scale

Figure 1. Procedural Plan for Study

A survey of attitudes towards the lesson was also made. A questionnaire was designed to measure the students' immediate reactions towards various lesson parts as well as his or her reaction to the overall lesson. The result was a five choice Likert-type attitude scale with both short answer and written response portions. The maximum positive response on the Likert scale has a value of 5 and a minimum value of 1.

Pilot testing of the attitude scale revealed that further in-depth responses to certain questions were needed, depending upon the students' answers to certain questions. These responses were obtained in a structured interview. The interview was individually administered and taped.

Each student who marked one or all of the designated questions on the attitude scale was interviewed to ascertain his or her free responses to those questions. The free responses provided detailed information with regard to the questions of concern.

The students who participated in the study were enrolled in physical science in a small ninth grade learning center during the school year 1978-79. The group included 16 males and 11 females. Students were randomly selected from two college-bound classes to participate in the study. These students had not been previously exposed to the lesson material.

The pretest was administered to the CAI students as a group. From one to ten days later they began the CAI lesson. The students were allowed to leave one of their regular classes to complete the CAI lesson. The lesson was given to individual students with a teacher available for aid and was normally completed in 45 minutes to 1 1/2 hours. After completion, about half of the participants were immediately given the post-test and attitude scale. The other half took the test the next day.

Results

The pre- and post-tests and attitude scale were hand scored. The statistical analyses were done on the Radio Shack TRS-80 microcomputer.

1. Were the content objectives for each part of the lesson fulfilled? The data indicate that 87.8% of the students fulfilled all of the 8 objectives.

2. Do students' pre- and post-test achievement scores indicate that they learned the CAI lesson material? Student achievement was analyzed in the four sections of the lesson: (1) electric circuit symbol identification, (2) electricity terminology, (3) Ohm's Law problems, and (4) circuit meter reading predictions. These results are summarized in Table 1.

3. What aspects of the CAI lesson were taught successfully? Four major categories of the test were examined in answering this question. The results are shown in Table 1. All students show improvement in all four areas. The largest gains in decreasing rank order are symbol identification, electric current, circuit meter readings, and Ohm's Law problems. The percentage of positive change ranged from 95.6% to 113.8%.

An examination of individual pre-test and post-test scores reveals the following data. The mean of the pre-test scores is 21.6% with a standard deviation of 18.32. The post-test mean is 93.83% with a standard deviation of 6.95. The largest increase in an individual student's test scores was 100% while the smallest increase was 22.33%.

The graph of the distribution of pre-test and post-test scores indicates two groupings (Figure 2). The pre-test scores are clustered at the lower end of the histogram. This indicates that without preparation, the material in the pre-test is too difficult for ninth grade students. High post-test scores reflect a substantial gain in student achievement levels. Table 2 shows that student achievement gain on all four parts of the test was statistically significant. The possibility of this gain occurring by chance ranges from 1 in 20 to 1 in 4,000.

4. What were the students' reactions toward the CAI lesson? Table 3 shows the results of the students' responses. Table 4 shows the

number of students who responded to each question on the Likert-type items. The percentage value is the number of students who responded to that question divided by the total number of students. Questions 2 and 6 were rated the most negatively. Question 2, dealing with the degree of lesson difficulty, did not affect the level of achievement attained. The negative response to question 6 was to be expected since 66.6% of the students had not had previous experience with a computer.

Table 1

Average Student Achievement on Each Category of Test

Test Cagetory	Pre test	Post test	△	△%	
Electric Symbol Identification	2.5	26.4	23.9	956.0	
Definition of Factors Affecting Electric Current	4.89	26.11	21.22	433.94	
Ohm's Law Problems	12	25.66	13.66	113.83	
Circuit Meter Readings	7.5	19.5	12	160.0	
Mean	6.72	24.42	17.69	420.94	
Variance	16.55	10.84	33.21	-	
Std. Dev.	4.07	3.29	5.76	-	
Paired-t-Test	-	df = 3	$t = 6.14073$	$P < .0025$	

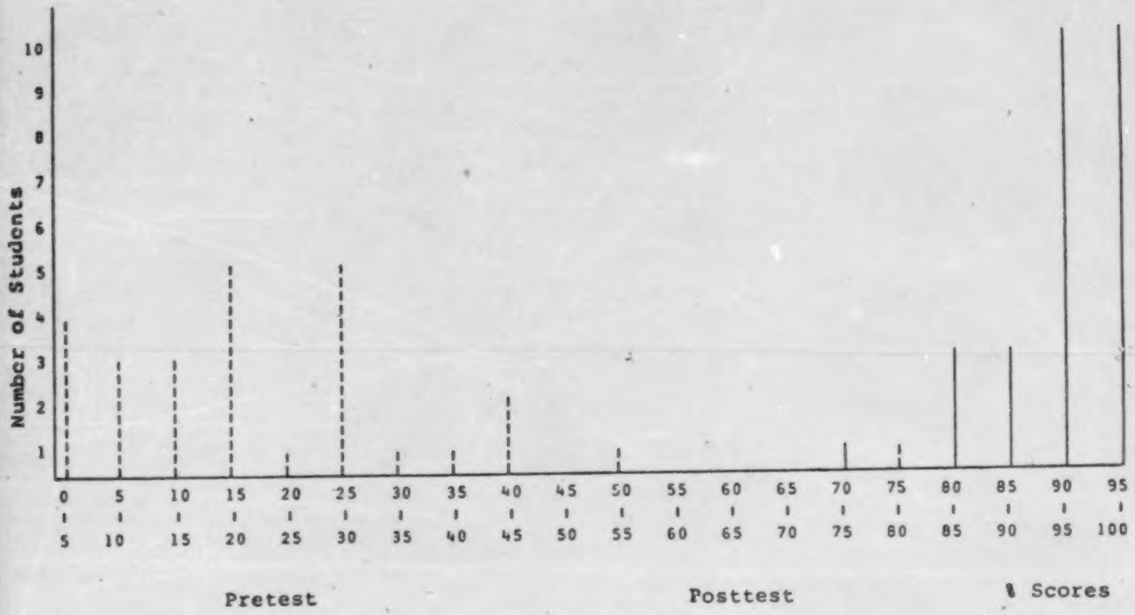


Figure 2. Pretest (-----) and Posttest (———) Score Distribution in Percent.

Table 2

Paired t-Statistic and Significance Level

Lesson Name	Paired-t-Test	Significance
Electric Symbol Identification	$t = 58.7878$	$p < .00025$
Definition of Factors Affecting Electric Current	$t = 11.5387$	$p < .00025$
Ohm's Law Problems	$t = 6.25244$	$p < .0125$
Circuit Meter Readings	$t = 4.0$	$p < .05$
Average Student Achievement on All Test Categories	$t = 6.14073$	$p < .0025$

Table 3

Student Attitude on Individual Questions from the Attitude Scale

Question No.	Mean	Median	Variance	Std. Dev.	Range
1	4.37	4	.32	.56	2
2	2.59	2	1.56	1.25	4
3	4.04	4	.42	.65	2
4	4.44	5	.72	.85	4
5	4.48	5	.41	.64	2
6	2.37	2	1.5	1.24	4
7	4.44	5	.72	.85	4
Average	3.81				

(5.0 Maximum positive value)

Table 4

Student Attitude Responses on Likert Scale Items in Percent

SA = Strongly Agree; A = Agree; NO = No opinion;
D = Disagree; SD = Strongly Disagree

	SA	A	NO	D	SD
1. I enjoyed the lesson.	40.7	55.5	3.7		
2. The lesson was difficult.	7.4	22.2	11.1	40.7	18.5
3. The practice problems were helpful in understanding Ohm's Law.	22.2	59.2	18.5		
4. The study guide was useful during the lesson.	55.5	40.7			3.7
5. The computer responses to my answers aided me is checking my progress.	55.5	3.7	7.4		
6. I had problems working with the computer equipment.	7.4	11.1	22.2	29.6	29.6
7. I prefer learning by the lecture method supplemented with the computer rather than strictly the lecture method.	55.5	40.7			3.7

The remaining attitude items and corresponding student responses follow:

ITEM 8. Have you had any previous experience in operating a computer?

Yes: 33.3% No: 66.6%

ITEM 9. There should have been (%)

	More	The Same	Less
a. Problems	51.9	48.1	
b. Circuit building	44.4	44.4	11.1
c. Computer assistance	33.3	66.6	
d. Lesson time	51.9	48.1	

ITEM 10. Which part of the lesson was boring?

Loading programs	9	(33.3%)
Progress quiz	3	(11.1%)
Building circuits	2	(7.4%)
None	1	(3.7%)
Symbols	1	(3.7%)
Ohm's Law problems	1	(3.7%)
Computer usage program	1	(3.7%)

The results from Item 10 were reasonable. With this computer's small memory capacity (four K bytes) it was necessary to write four different programs corresponding to the four lesson parts. As each part was needed it had to be loaded into the computer. The loading time for the cassette recorder is about two minutes per program, and this waiting time bores students.

ITEM 11. Which part of the lesson was interesting?

All	15	(55.5%)
Circuit building	7	(25.9%)
Progress quiz	3	(11.1%)
Computer responses	1	(3.7%)
Ohm's Law Problems	1	(3.7%)

ITEM 12. Was the lesson confusing?

Yes	4	(14.8%)
At times	10	(37%)
No	13	(14.1%)

ITEM 13. What did you like most about using the computer?

No answer	7	(25.9%)
Using the computer	6	(22.2%)
Computer responses	5	(18.5%)
It was different	3	(11.1%)
All	2	(7.4%)
Its memory	1	(3.7%)
Easy	1	(3.78%)
Fun	1	(3.7%)
Progress quiz	1	(3.7%)

ITEM 14. What did you dislike most about using the computer?

Nothing	11	(40.7%)
No answer	9	(33.3%)
Loading the program	5	(18.5%)
Progress quiz	1	(3.7%)
More explanations	1	(3.7%)

ITEM 15. I would like to use the computer to learn 75.4% of the time.

ITEM 16. Did the computer lesson help you to learn the topic?

Yes	26	(96.3%)
No	1	(3.7%)

(Oddly, the student who answered no to this question answered 100% for question #15 above.)

Additional student comments included:

"It was a blast!!!!!!"
"I would like to do it more."
"It would be nice if we're able to use the computer this way more often."
"I liked it a lot. It made learning fun."
"This method is very effective!!"
"The computer tape recorder would not cut off properly. Please have it repaired."
"It was fun."

The results of the interviews were as follows:

ITEM 6. I had problems working with the computer equipment.

"Yes, the U key made multiple U's."
"Yes, I couldn't go back in the lesson."

ITEM 7. I prefer learning by the lecture method supplemented with the computer rather than strictly the lecture method.

"Yes, with the lecture method I can ask questions at any time."

ITEM 12. Was the lesson confusing?

"Yes, there could have been more explanations on how to build the circuits."

ITEM 16. Did the computer lesson help you to learn the topic?

"It helped with all except the circuit building."

5. Was there a relationship between student achievement scores and student reactions toward the CAI lesson? The Pearson product-moment correlation coefficient was used to determine if any relationship existed between the student's post-test score and his attitude score; and, the student change in score from pretest to post-test and his attitude score. Post-test Score to Attitudes Score Correlation Coefficient = .226; Pre and Post-test Score Change to attitude Score Correlation Coefficient = -.262. This indicated an undefined relationship between achievement and attitude in this study.

Conclusion

The evidence from this study supports the conclusion that computer-assisted instruction can effectively increase student achievement in secondary physical science while leaving the student with a positive attitude toward the learning experience.

Recommendations for Further Study

This research suggests a number of areas for further study in this new field such as:

1. In what content ares and in what ways can CAI be most effective in supplementing traditional instruction?
2. Specifically how does CAI improve instruction?
3. What are the long-term effects of achievement gain through CAI?
4. How are student attitudes toward CAI affected over the long-term?
5. How can the design and development of CAI programs be improved?
6. How can the development of quality CAI programs be encouraged?
7. What are the most effective means of introducing educators and administrators to the benefits and limitations of CAI?

REFERENCES

Bloom, B.S. (ed.) 1956. A Taxonomy of Education Objectives: Cognitive Domain. New York: Longmans Green.

Bukoski, William J. and Korotkin, Arthur L. 1976. Computing Activities in Secondary Education. Educational Technology. Jan.: 9-23.

Feldhusen, John and Szabo, Michael. 1969. The Advent of the Educational Heart Transplant, Computer Assisted Instruction: A Brief Review of Research. Contemporary Education. 40: 265-274.

Glaser, R. (ed.) 1965. Teaching Machines and Programmed Learning, II, Data and Directions. Washington, D.C.: Department of Audio-Visual Instruction, National Education Association.

Skinner, B. F. 1968. The Technology of Teaching. New York: Appleton-Century-Crafts.

Zinn, Karl L. 1978. A Place for Personal Computing in Schools and Colleges. Interface Age. 3: 70-75.

AN OVERVIEW OF MICRO-CMI

Sponsored by AEDS

Don McIsaac
Educational Sciences Building
University of Wisconsin-Madison
1025 West Johnson Street
Madison, WI 53706
(608) 263-2718

Abstract:

MICRO-CMI is a computer program developed for the LSI-11 computer. It is the third generation of CMI development following implementation of a similar kind of program on a Univac 1100 time-sharing system at the University of Wisconsin. MICRO-CMI supports 1000 students in 37 different program areas, maintaining up to 2500 grade records in each curriculum. The program groups students according to instructional needs when asked to do so, and will provide a diagnosis and prescription if these data have been entered into the data base. The program currently operates in Milwaukee and McFarland, Wisconsin, Danville, Illinois, and Lake Stevens, Washington.

MICRO-CMI demonstrates the viability of microcomputers in the schools, performing the important classroom tasks that emerge with an individualized program. These individualized programs require extensive data collection and report generation. MICRO-CMI is designed to perform that need in an efficient and cost-effective manner.

NETWORK INFORMATION RESOURCES FOR COMPUTERS IN TEACHING AND LEARNING

Chaired by Karl Zinn
University of Michigan
Ann Arbor, MI 48109

ABSTRACT:

Many people are seeking current and accurate information about computer uses in teaching and learning. The burden on advisors is increasing rapidly due to inexpensive computers, advertising by vendors, inclusion of computers in technical curricula, and recognition of impact of computers on careers. Although the sources of good information may continue to be national projects, centers, clearinghouses, publishers, and vendors, the delivery and interpretation will be done more effectively through local resource people (librarians in media or learning resource centers, staff in science centers, curriculum coordinators, etc.).

This session sums up what is being done by public agencies, information centers and projects, professional associations, commercial organizations, R & D projects, and college or university programs to help local resource people. It will serve two purposes: provide information for those new to the field; and hold discussion among those responsible for providing information to teachers using computing. For example, can the technology be useful in dissemination (interactive decision aids, computer-controlled video, directories in computer-readable form, directories on national data services, community bulletin board systems, electronic mail, local adaption of computer-based directories, etc.)? What models do we have for effective support networks? What approaches or techniques have already shown some success?

PARTICIPANTS:

James Johnson
CONDUIT
University of Iowa
Lindquist Center
Iowa City, IA 52242

Joseph Lipson
National Science Foundation
1800 G Street N.W.
Washington, DC 20550

Laurette Lipson
National Medical Audio Video Center
8600 Rockville Pike
Bethesda, MD 20209

PRECOLLEGE COMPUTER LITERACY

Ronald E. Anderson
Edward E. Wachtel

ABSTRACT: Instructional Materials for Computer Literacy

Ronald E. Anderson, Computer Literacy Project, Minnesota Educational Computing Consortium, 2520 Broadway Drive, Saint Paul, MN 55113

With funding from the National Science Foundation, the Minnesota Educational Computing Consortium (MECC) will design, develop, test, and evaluate an integrated set of twenty-five student learning modules for science, mathematics, and social studies courses in secondary schools. The central theme of these learning packages will be the computer -- how it works, how it is used, its impact on the individual and society, and the implications of its use for the future. The project presumes that abilities, skills, attitudes, and cognitive learning are all aspects of a person's computer literacy; the learning packages will be based on a set of learning objectives that reflect these areas. The materials will be written for students entering junior high school. However, they could be used at other secondary school levels as well.

ABSTRACT: Computer Science Summer School Offering: A Place to Start

Edward E. Wachtel, Computer Education Steering Committee, Shaker Heights City School District, 15600 Parkland Drive, Shaker Heights, OH 44120

The Shaker Heights City Schools offered a non-credit junior high computer course in the summer of 1980 for a limited number of students. It was a pilot for computer science offerings to be developed during the 1980-1981 school year.

The course included thorough instruction in the Basic programming language on a variety of hardware, two field trips, and discussions of computer-related topics.

The structuring of the course and its eventual execution and evaluation has given valuable information for the further development of credit computer science offerings in the Shaker schools. It has also spurred the possible creation of future summer computer science offerings and similar offerings open to adults through the local recreation board.

ABSTRACT: A Five-Year Time Line for the Integration of Computers into the Shaker Heights City Schools

Edward E. Wachtel, Computer Education Steering Committee, Shaker Heights City School District, 15600 Parkland Drive, Shaker Heights, OH 44120

The Shaker Heights City Schools, a suburban district of approximately 6000 students, funded a project in the summer of 1980 to prepare a detailed five-year plan for integrating computers into the schools, K through 12. Such a plan was developed for presentation by 1 September 1980 by a team within the district and included explicit recommendations for hardware and software purchases, the expansion of CAI-CMI, the establishment of a computer science department and consequent computer science offerings at the secondary level, administrative structure, staff training structures, estimated costs and other related considerations.

A discussion of the plan and its creation would be of value to others contemplating a major move to computers in their school systems. By the

presentation some critical observations concerning the implementation of this plan will also be possible.

REPORT OF ACTIVITIES OF THE ACM
ELEMENTARY AND SECONDARY SCHOOLS SUBCOMMITTEE

Co-Chaired by

J. Philip East
Computer & Information Science
University of Oregon
Eugene, OR 97403

Jean B. Rogers
Computer & Information Science
University of Oregon
Eugene, OR 97403

ABSTRACT:

The ACM Elementary and Secondary Schools Subcommittee (ES3) of the ACM Curriculum Committee on Computer Science was formed in June, 1978. Originally, twenty-nine task groups were organized to investigate and report on specific aspects of computer use and instruction in pre-college education; they gathered and organized large amounts of information and started producing working papers and preliminary reports.

By the summer of 1980, several groups had completed their tasks, and others had interm reports written. In January 1981, the Education Board of ACM, along with the ACM Special Interest Groups in Computer Science Education (SIGCSE) and Computer Uses in Education (SIGCUE), published reports by seventeen task groups in a special bulletin called TOPICS Computer Education For Elementary and Secondary Schools.

Two final reports included in this publication have been accepted by the ACM Education Board as statements of the board's position on the matters discussed. One of these reports is "Teacher Education" by Jim Poirot, Robert Taylor, and Jim Powell. This report specifies competencies needed by all teachers at these levels, whatever their primary discipline might be, as well as competencies needed by teachers of computing. A second final report is "Computer Science in Secondary Schools: Recommendations for a One-Year Course" by Jean Rogers and Dick Austing. This report suggests some factors to consider in implementing a secondary school level computer science course, as well as delineates the study topics to be covered in such a course.

Summaries of the two final reports, the report on Ethical and Social Concerns and the report on Programs for the Talented and Gifted, will be presented. Other reports may also be discussed, depending on the preference of those people present.

Another portion of this report session will be devoted to a review of the past and present activities of ES3. This discussion will include the original list of task groups and its evolution, an enumeration of task groups that produced reports, and a discussion of the process of developing and approving reports.

ES3 is currently undergoing both structural and procedural changes. Two specific questions will be considered in this context:

1. Which of the tasks that have been identified should be given emphasis in the near future?

2. What procedure should be used to develop future reports?

A major function of this presentation is to allow discussion of these issues.

Session attendees will be provided the opportunity and are encouraged to comment on the reports and other aspects of ES3.

THE FUTURE OF MICROCOMPUTERS IN EDUCATION: A VENDOR'S VIEW

Glenn Polin
Apple Computer, Inc.
Cupertino, CA 95014

ABSTRACT:

How does the world of microcomputers in education look to one of its principal manufacturers, Apple Computer, Inc.? This session will focus on the expanding world of microcomputers in computer literacy, programming education, and CAI. The role of the major education publishers (such as SRA) and major user groups (such as MECC) in the production of CAI will be discussed; also, the economies of scale required for widespread CAI development and usage will be described.

PERSONAL COMPUTING TO AID THE HANDICAPPED:
THE JOHN HOPKINS FIRST NATIONAL SEARCH

Sponsored by IEEE/Computer Science

Paul L. Hazan
The Johns Hopkins University
Applied Physics Laboratory
Johns Hopkins Road
Laurel, MD 20810

ABSTRACT:

Extending the reach and improving the quality of life of disabled people are surely among the most worthwhile achievements that will result from the expanding use of computers. The promising alliance between the power of computing technology and the urgent needs of the handicapped is providing opportunities that are capturing the imagination of both the handicapped and the computer communities.

Sympathizing with the need to relieve many constraints under which handicapped people live and to turn opportunities into reality, the Johns Hopkins First National Search was conceived as a nation-wide competition to inspire computer-based applications aimed at meeting the daily living, educational, and vocational needs of disabled persons.

The objectives of the competition are to:

Focus the power of computing technology on the urgent needs of handicapped citizens.

Foster individual innovation and creativity throughout the nation.

Encourage individuals, professional societies, and academic, industrial, government, civic, and rehabilitation organizations to work together to meet a major national need.

Since the ultimate objective is to get solutions into the hands of those who need them, a nationally publicized two-day workshop will be held at the conclusion of the program to bring together search participants, potential users, government agencies, industry, academic and rehabilitation institutions. To recognize and further publicize the achievements of the winners, proceedings describing the winning submissions will be published.

The Johns Hopkins National Search is an opportunity for the imaginative to help realize the claimed potential of this new technology. Much has been said about the potential of personal computing for the handicapped, and a number of individual successes have been achieved. For most of us, however, this is an exceptional opportunity to translate ideas into reality.

A ten minute video tape highlighting applications of Personal Computing to Aid the Handicapped was part of this conference session.

THE STUDY OF U.S. REGIONAL SHIFTS, METROPOLITAN GROWTH AND NEIGHBORHOOD STRUCTURE WITH CENSUS MICRODATA*

Harold Benenson
Visiting Assistant Professor
Sociology and Urban Studies
Vassar College
Box 512
Poughkeepsie, NY 12601

Steven Just
Mathematica Policy Research
P.O. Box 2393
Princeton, NJ 08540

INTRODUCTION

In a previous paper (Benenson and Just, 1979) we discussed the advantages of instructional use of computerized census data as an alternative to two prevailing approaches in the sociology research methods curriculum. The latter consist of formulations of research projects involving, student-gathered data, and, pre-packaged survey data sets. We have found the central weakness of the student-generated survey is that the resulting data set is often trivial. The superficial scope and depth is caused by limitations of time, survey resources, and student inexperience. The other widely used approach, using the pre-packaged survey, provides a well developed data set; however the content of the data is often remote. In addition the attitudinal focus of most available surveys limits their relevance for the analysis of structural dimensions of change, for example in economic, demographic, family, residential, and other domains. But it is precisely this type of structural analysis which is increasingly critical in applied cial research outside the university.

To surmount these problems, we turned to the computerized 1970 Census *Public Use Samples of Basic Records (PUS)* as a data source for student analysis. We obtained a PUS file of 95,000 records for New Jersey and had students examine of household and person records for two or more counties. A typical assignment in a senior level sociology seminar on "Research Methods in Community Analysis" asked students to create comparative socio-economic and demographic files of two counties. Students analyzed the labor force, racial, income, household type, and home ownership characteristics of the counties, concentrating on contrasting patterns, the causes of these patterns, and the development of questions for further research.

Our innovations beyond this initial stage have led us to reorient the instructional census data to address issues of national scope, package the data in a form that will be useful to faculty in sociology, urban studies, and applied social sciences in all college settings, and develop new foci for student research.

DEVELOPMENT OF THE PROJECT

The major new challenges confronting our project's work with computerized census data have been posed by the instructional context of the Urban Studies program at Vassar College. In this setting we have attempted to develop instructional materials that address problems of urban development and policy in diverse geographic environments, and that exploit the real world frame of reference (and advantages) of these data as effectively as possible. New objectives in the instructional use of urban census data have been clarified in three areas:

(1) *The theoretical goal of clearly defining research issues in relation to processes of metropolitan decline and growth and the class and ethnic structure of cities*: Our earlier work had formulated questions for student research with census data to develop profiles of New Jersey counties with emphasis on describing the interaction among economic, racial, and sexual dimensions of stratification for these areas. Our recent work has focused research uses of the *PUS* around critical issues of urban change that have national significance. Urban sociologists, demographers, and economists have discussed extensively the uses (and anadequacy) of census data (especially the 1980 Census) in relation to the urban poor (Exum et al, 1980; U.S. Bureau of the Census, 1980); the causes of urban declin and crisis in the Northeast (Sternlieb and Hughes, 1975); expansion in the sunbelt (Perry and Watkins, 1977); the changing

class and ethnic structure of cities, and the relation of urban poverty and male and female employment to changing household and family patterns (Clark and Gershman, 1980; Stack, 1974; Ross and Sawhill, 1975). Our new instructional data sets, titled Metropolis, Milltown and Sunbelt City: Comparative Census Data for Urban Residents, links each substantive research problem to a particular type of urban community (or metropolitan area), and then to a specific Census PUS file created for a community of that type.

(2) The methodological goal of selecting the appropriate geographic units, the specific communities for study, and a mode of data organization to realize this theoretical objective: Our earlier geographic focus was exclusively county units within a single state, and involved study of rural (southern New Jersey) as well as urban and suburban (northern New Jersey) county patterns. The new data sets examine two geographic levels (the urban neighborhood and the Standard Metropolitan Statistical Area [SMSA]), contain data for communities in diverse (Middle Atlantic, East North Central, and Mountain) regions of the United States, and organize data sets to highlight comparative analysis of distinct types of urban social structure and dynamics. The neighborhood-based data sets include three communities in fairly close proximity within a major metropolitan center that provide contrasts in class structure, employment and income, ethnic composition, and housing characteristics and family structure. The SMSA data set compares one medium-sized industrial center in the Northeast that has experienced severe economic crisis and population decline, with an expanding SMSA in the Southwestern sunbelt that has undergone rapid population and economic growth.

(3) The technical goal of producing medium-sized data sets for use with SPSS: Our original census data set was both large (95,000 records of 120 characters each, in a hierarchical structure) and assumed the availability of special-purpose software (CENTS-AID) and Data Base Dictionaries (described in DUALabs, 1976; Benenson and Just, 1979). To increase ease of use and adaptability, the Metropolis, Milltown and Sunbelt City materials consist of a series of data sets of medium size (5,000 to 9,000 records apiece) in SPSS System File format (Nie et al. 1975; 78-89). The latter combine data files that include variables from the original Census PUS (30-40 in all) and additional computer-generated variables, with labels for variables and subfiles.

Our conception of the Metropolis, Milltown and Sunbelt City data sets attempts to realize these three objectives, and fill a gap in the availability of computerized materials for the study of metropolitan development, regional shifts, and urban stratification patterns in contemporary American society (Anderson, 1977). We were familiar with the valuable historical Comparative Cities II SPSS System File-data set (Litchfield, 1978) that contains census records for mid-19th century Pisa (Italy), Amiens (France), Stockport (England), and Providence (Rhode Island, U.S.), and orients student analysis toward questions of urban growth, industrialization, and household structure and employment patterns. We analyzed primary census sources for comparable current urban issues within a dynamic perspective.

CREATING THE DATA SETS

In developing the neighborhood-centered Metropolis segments of our materials, we made use of little-known special tabulations of 1970 Census Public Use Sample records for New York City neighborhoods. The conventional 1970 com-Census Public Use Samples of Basic Records (PUS) are six national data files containing approximately 2 million records for persons and half-a-million records for households each. The PUS were created from the computerized facsimiles of two kinds of census questionnaires: one long form sent to 15 percent of households, and another sent to 5 percent of households. The two forms include slightly different combinations of questions concerning basic demographic, ethnic, educational, employment, income, family structure, and other personal as well as housing unit characteristics (U.S. Bureau of the Census, 1972a). The PUS represent "microdata" (i.e., data organized in relation to individual persons and households) from the decennial censuses in contrast to the detailed aggregated data available in Census Summary Tape Files. The major limitation of the PUS for urban research has been that researchers cannot analyze person (or household) records for units smaller than counties.

The availability of special tabulations of 1970 PUS organized by identifiable neighborhoods (for New York City and Chicago) remedies this limitation for two cities (U.S. Bureau of the Census, 1979: 44-45). The 1980 Census PUS which will permit geographic identification of microdata in relation to areas containing 100,000 population, in contrast to the limitation of 250,000 for the 1970 PUS, and will vastly expand research possibilities with local-level microdata [Zeisset et al, 1981]. It will also make possible the future creation of time-series data sets for our Metropolis N.Y.C. neighborhoods.

The N.Y.C. special tabulations, which identify 27 neighborhoods within the city, were created to order for the N.Y. Department of Commerce in 1974 as part of a special tabulation project costing approximately $6,900. The two tabulations (one based on the 5% PUS questionnaire, the other, on the 15% questionnaire) have been made available by the Census Bureau to any user for $160. For the Metropolis data sets we selected three neighborhoods fairly close together--the South Bronx, the East Side and West Side of Central Park in Manhattan (from 82nd Street to 110th Street), and Astoria-Long Island City in Queens (see Figure I)--that cover an extreme range of income levels and class and ethnic composition patterns. For example, the average level of family income for individual census tracts in these areas ranged from less than $5,000 for certain parts of the South Bronx to more than $45,000 for Park Avenue-Fifth Avenue Tracts in the 80s blocks of Manhattan (U.S. Bureau of the Census, 1972b p206, p208). Comparative characteristics for residents of the three neighborhoods are presented in Table 1. Neighborhood boundaries were predetermined by the original special tabulations, and especially in the case of the Manhattan neighborhood, encompassed socially diverse populations in the single area. To help analyze distinct groups within a neighborhood, income level subfiles (grouping persons by household income levels of $50,000 or more, $25,000-49,999, $15,000-24,999 and less than $15,000) were created, and for the Manhattan area, the sample size was increased from 1 percent of the population to 2 percent (by combining the 5% and 15% PUS into a single data-set). The organization of the two Metropolis data sets are described in Table 2. In schematic terms, the following social types of communities can be analyzed and compared with this data:

A) South Bronx. An extremely poor ghetto community, composed of Hispanic (largely Puerto Rican) as well as some black residents.

B) Astoria-Long Island City, Queens. A white working class (and lower-middle class) community, with a strong concentration of residents of immigrant (primarily Italian) background.

C) Central Park, East Side and West Side, Manhattan. Primarily an affluent upper-middle class community, based in professional and managerial occupations, containing an extremely wealthy upper-class enclave and a segment of Spanish Harlem (East 96th Street-110th Street).

The task of creating the Milltown-Sunbelt City data set was more straight-forward. We selected medium-sized SMSA's (with central city populations of 100,000-250,000 in 1970) in the industrial East North Central region (Youngstown, Ohio) and the sunbelt of the Southwest (Albuquerque, New Mexico). These SMSA's exemplified on a small scale the patterns of metropolitan decline and expansion, and economic and demographic change of their regions. The 1970 Census data depict these cities at a midpoint of recent development. Table 3 summarizes some major characteristics of published Census reports. It highlights, in particular, the contrast between the industrial, immigrant, long-term resident background of many Youngstown residents, and the different and internally heterogeneous backgrounds of the Albuquerque population. Such contrasts persisted and widened during the 1970-1980 decade. In this period the former SMSA experienced major economic crisis with the closing of steelmills and lost 17 percent of its 1970 population, while Albuquerque (like the larger Southwestern cities of Phoenix and San Jose) grew by 35 percent ("Population Shift," 1980; U.S. Bureau of the Census, unpublished preliminary 1980 Census counts, January 1981). The analysis of the Milltown-Sunbelt 1970 microdata enables students to examine the position and experience of varied demographic and economic groups in the midst of urban change. The 1970 analyses can also serve as a baseline for understanding the social impact of 1970-1980 developments.

The characteristics of this data set are described in Table 2. It was created from two conventional 1970 Census Public Use Sample tapes (one percent sample, 15 percent questionnaire form) which cost $80 apiece from the Census Bureau (Customer Services Branch).

A considerable amount of pre-processing of raw census files was necessary to generate a format for the data sets that could be used with commonly available software (e.g., SPSS). The major task involved extracting person records and eliminating household records to "rectangularize" the data sets. In addition, person records in all data sets were sorted on the basis of the higher level (i.e., household record) variable of household income level.

Use in the Classroom

The Metropolis, Milltown and Sunbelt City microdata are being developed and used experimentally in the interdisciplinary Urban Studies program at Vassar College. Students run SPSS programs using the Metropolis data sets, analyze substantive issues of race, class, employment, and family structure in New York City neighborhoods, and critique alternative uses of computerized census data in this required course. In addition, we have discussed incorporating similar study projects into a new course offering on the Sociology of Residential Neighborhoods.

Students who use the data sets usually have some background in statistics and introductory work with the computer. Most take the Urban Research Methods course in their junior year. Using census microdata for urban areas (New York City neighborhoods) that many of the students are familiar with has been very well received. Students with applied research interests (prospective planners, architects, social workers), and those with a keen interest in social justice and the city have reacted particulary favorably to the opportunity of using their skills with this type of datum and research problem.

From the standpoint of teaching, the major difficulties and challenges center on integrating census data anlaysis with broader theoretical approaches and critical modes of understanding. In Urban Research Methods students are exposed to major urban policy analyses which have relied on government statistics, such as the well-known Moynihan Report on the Black family (Rainwater and Yancey, 1967); specific critiques of this analysis based on qualitative (participant observation) research methodologies (Stack, 1974); and general discussions of the biases of official social statistics and their computerized foundation (Hindess, 1979; Irvine et al, 1979). Students often respond to these controversies, and to the difficulties of their own statistical research, by shifting from a naive believe in all statistically derived "facts" to an equally naive opposition to anything quantitative.

For theoretical perspective, the Research Methods course integrates computerized analysis of census microdata with other quantiative and qualitative research exercises--for example, participant studies of community class structure undertaken in relation to major theories of class, exercises with published census tract data for the same communities, comparisons between the types of raw data (for persons and households in specific neighborhoods) derived from different methodologies and theoretical perspectives, and so on.

The course attempts to probe the limitations of each research strategy (including strategies using computerized census data) by discussing the varied theoretical frameworks which have informed urban research.

CONCLUSIONS

Using census data as described in this project represents a significant departure from our previous work. We have reoriented our focus from using data sets with limited local interest to creating data sets which are national. The expansion and decline of urban centers is a topic of current interest to many social scientists. The "Metropolis, Milltown and Sunbelt City" data sets permit students to gain information as current as the most recent census.

Our plans call for disseminating the data sets to a wider user community and creating parallel data sets from the 1980 Census to facilitate longitudinal studies.

---

*The development of the Metropolis, Milltown and Sunbelt Cities Data Sets, described in this paper, were made possible by a Mellon grant at Vassar College and technical support from the New Jersey Educational Computer Network. Research assistance was provided by Ms. Virginia Bronzi.

REFERENCES

Anderson, Ronald et al (1977). "Computers in Teaching Sociology." In *Computers in Undergraduate Teaching*. Iowa City, Iowa: The University of Iowa.

Benenson, Harold (1979). *Computer Analysis of Census and Government Research Data and Introduction to CENTS-AID, with Appendices on Using SPSS with the 1970 Census Public Use Samples*. New Brunswick, NJ: New Jersey Educational Computer Network.

Benenson, Harold and Steven Just (1979). "The Use of Census Data Analysis to Enhance the Sociology Curriculum." In Diana Harris (ed.) *Proceedings of National Educational Computing Conference - 1979*. Iowa City, Iowa: The University of Iowa.

Clark, Kenneth and Carl Gersham (1980). "The Black Plight: Race or Class?" *New York Times Magazine*, October 5, 1980.

DUALabs (1976). *CENTS-AID II User's Manual*. Arlington, VA: DUALabs

Exum, William et al (1980). "The Census Undercount," In *Black People and the 1980 Census*. Chicago: Illinois Council for Black Studies.

Hindess, Barry (1979). *The Use of Official Statistics in Sociology*. London: MacMillan.

Irvine, John et al (1979). *Demystifying Social Statistics*. London: Pluto.

Litchfield, R.B. (1978). "Comparative Cities II--Course Book." Providence, RI: Brown University.

Nie, Norman et al (1975). *SPSS*. New York: McGraw-Hill.

Perry, David and Alfred Watkins (1977). *The Rise of the Sunbelt Cities*. Beverly Hills, CA: Sage.

"Population Shift Puts New Cities in Top 10" (1980). *The New York Times* (December 19).

Rainwater, Lee and William Yancey (eds.) (1967). *The Moynihan Report and the Politics of Controversy*. Cambridge, MA: MIT Press.

Rose, Heather and Isabell Sawhill (1975). Time of Transition: *The Growth of Families Headed by Women*, Washington, D.C.: The Uran Institute.

Stack, Carol (1974). *All Our Kin*. New York: Harper and Row.

Sternlieb, George and James Hughes (1975). *Post-Industrial America: Metropolitan Decline and Inter-regional Job Shifts*, New Brunswick, Rutgers Center for Urban Policy Research.

U.S. Bureau of the Census (1972a). *Public Use Samples of Basic Records from the 1970 Census: Description and Technical Documentation*. Washington, D.C.: Bureau of the Census.

U.S. Bureau of the Census (1972b). *Census of Population and Housing: 1970. Census Tracts. Final Report. PC (1)-145. New York, NY SMSA*. Washington, D.C.: U.S. Government Printing Office

U.S. Bureau of the Census (1979). *Directory of Data Files*. Washington, D.C.: Bureau of the Census.

U.S. Bureau of the Census (1980). *Conference on Census Undercount*. Washington, D.C.: U.S. Government Printing Office.

Zeisset, Paul et al (1981). "Public Use Sample Microdata Background paper." Washington, D.C.: U.S. Bureau of the Census.

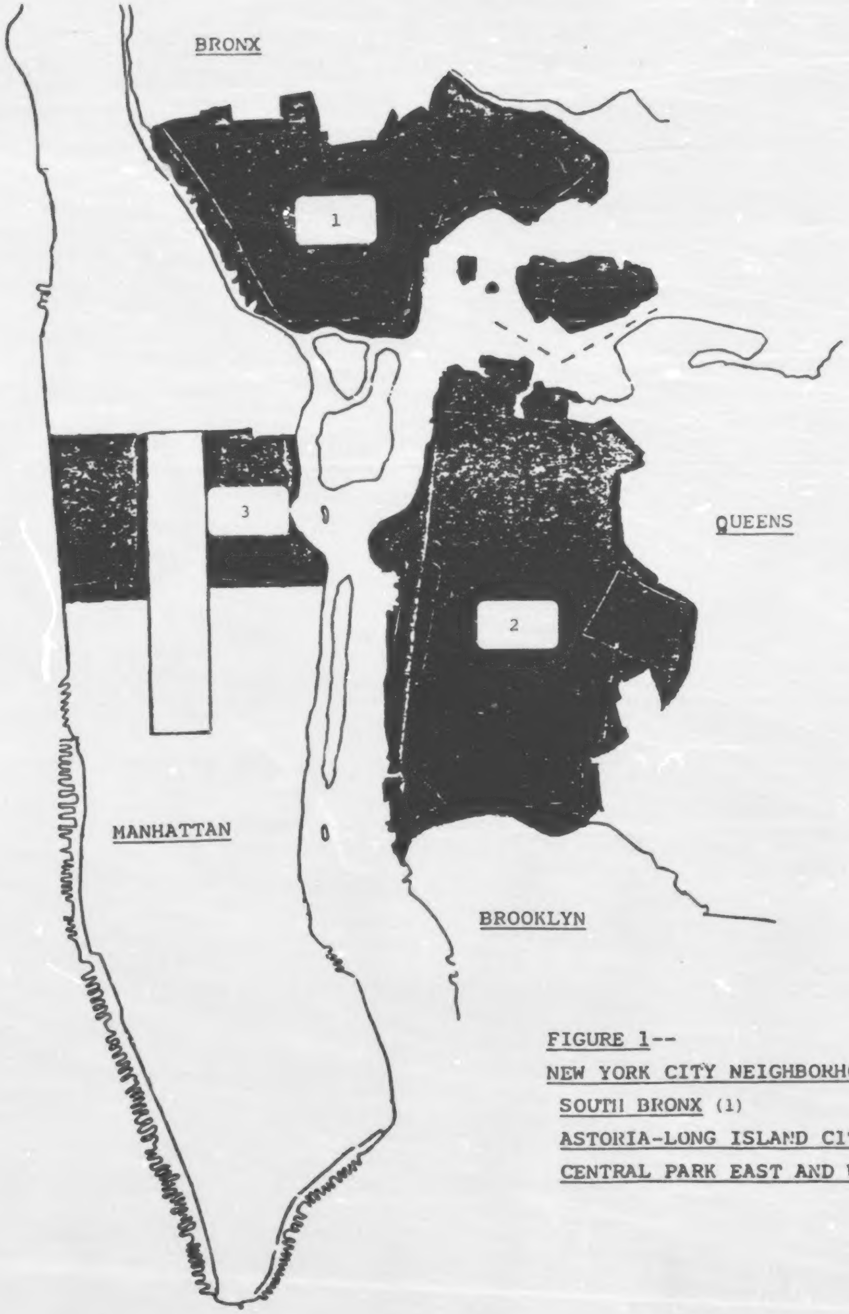

BRONX
1
3
2
QUEENS
MANHATTAN
BROOKLYN

FIGURE 1--
NEW YORK CITY NEIGHBORHOODS
SOUTH BRONX (1)
ASTORIA-LONG ISLAND CITY (2)
CENTRAL PARK EAST AND WEST (3)

TABLE 1

Characteristics of Residents	South Bronx	Astoria-L.I.C. (Queens)	Central Park East-West (Manhattan)
POPULATION & ETHNICITY			
Total Population	251,855	272,161	260,531
Percent Black	35.2%	4.9%	17.9%
Percent Hispanic[a]	55.0%	10.5%	26.9%
Percent Foreign Born or of Foreign Parentage	16.8%	61.1%	37.5%
FAMILY			
Percent of families supported by unmarried female[b]	30.1%	14.0%	26.3%
EMPLOYMENT			
Percent of employed persons in professional occupations	6.9%	9.7%	23.6%
Percent of employed persons in Blue Collar occupations	42.1%	31.7%	15.6%
INCOME			
Average Family Income	$ 7,030	$10,537	$16,891
Percent of families receiving public assistance	26.9%	4.3%	11.3%

---

[a] Percent "Persons of Spanish Language."

[b] Percent "Families with Female Head."

SOURCE: U.S. Bureau of the Census. Census of Population and Housing: 1970. CENSUS TRACTS. Final Report PHC (1) - 145 New York, NY SMSA Parts 1,2. Washington, D.C.: U.S. Government Printing Office, 1972. Tables P-1, P-2, P-3, P-4.

TABLE 2

CONTENT AND STRUCTURE OF SPSS "METROPOLIS, MILLTOWN AND SUNBELT CITY" - SYSTEM FILES

System File	Source of Data	Geographic Unit & Size	Approx. No. of Records and Sample Size	No. of Variables	Subfile Structure	Central Research Focus
Manhattan-Central Park East and West neighborhood	5% questionnaire and 15% questionaire 1970 Census Public Use Samples-N.Y.C. special tabluations[a]	Neighborhood-(Manhattan) approx. 2 sq. mi. (37 Census Tracts)[d]	5200 1/50 sample	35[f]	Person records for households with incomes of: (a) $50,000 or more (b) $25,000-49,999 (c) $15,000-24,999 (d) less than $15,000	Economic-class divisions within a neighborhood
Queens (Astoria-Long Island City and Bronx (South Bronx) neighborhoods	15% 1970 Census PUS - N.Y.C. special tabulation[b]	Neighborhood-Astoria L.I.C., Queens = 11 sq. mi. (88 tracts) South Bronx = approx. 9 sq. mi. (51 tracts)[d]	5200 1/100 (2700 records) (2500 records)	45	Astoria-L.I.C residents by household income groups (a), (b), (c), (d). South Bronx residents by groups (a), (b), (c), (d).	Neighborhood in class, ethnicity, employment and family patterns
Youngstown-Albuquerque standard metropolitan statistical areas	15% 1970 Census PUS - national 1/100 County Group sample[c]	SMSA Youngstown = 1030 sq. mi. (Mahoning & Trumbull counties) Albuquerque = 1169 sq. mi. (Bernalillo County)[e]	8600 1/100 (5400 records) (3200 records)	45	Youngstown residents by household income groups (a), (b), (c), (d) Albuquerque residents by groups (a), (b), (c), (d)	Urban change and metropolitan differences in employment and income, mobility and demographic structure

---

[a] U.S. Bureau of the Census (1979) Directory of Data Files, Washington, D.C.: Bureau of the Census, p. 44-45. Data Files "C CEN 70 035" (5% questionnaire, N.Y.C. special tabulation) and "C CEN 70 031" (15% special tabulation).

[b] See note (a). Data File "C CEN 70 031" only.

[c] U.S. Bureau of the Census (1972) Public Use Samples of Basic Records From the 1970 Census Description and Technical Documentation. Washington, D.C.: Bureau of the Census, p. 15. Data Files "PUS70203" (including Youngstown SMSA, county group no. 902) and "PUS 70225" (including Albuquerque SMSA, no. 12901).

[d] Based on Data File "C CEN 70 031" documentation.

[e] U.S. Bureau of the Census (1973) City and County Data Book, 1972. Washington, D.C.: U.S. Government Printing Office. p. 548, 578.

[f] Includes only variables common to 1970 Census "5 percent" and "15 percent" questionnaire formats.

TABLE 3

COMPARATIVE CHARACTERISTICS OF METROPOLITAN AREA RESIDENTS--
"MILLTOWN" (YOUNGSTOWN SMSA) AND "SUNBELT CITY" (ALBUQUERQUE SMSA)

Characteristics of Residents	Youngstown, Ohio SMSA	Albuquerque, New Mexico, SMSA
POPULATION & ETHNICITY		
Total Population	536,004	315,774
Percent Black	9.4%	2.1%
Percent Hispanic[a]	1.1%	37.0%
Percent Foreign Born or of Foreign Parentage	21.8%	8.5%
FAMILY & EDUCATION		
Percent of Families Supported by Unmarried Female[b]	9.8%	12.0%
Percent High School Graduates	52.1%	66.2%
EMPLOYMENT		
Percent of Employed Persons in Professional Occupations	11.3%	22.0%
Percent of Employed Persons in Blue Collar Occupations	47.6%	25.0%
Percent of Employed Persons in Manufacturing Industry	42.8%	7.5%
INCOME		
Average Family Income	$11,411	$10,370
Percent of Families with Incomes below Poverty Level	6.5%	13.0%
MIGRATION & HOUSING		
Precent of Population moved into SMSA since 1965	7.9%	22.1%
Percent of Occupied Housing Units which are Owner Occupied	75.3%	65.2%

[a] Percent "Persons of Spanish Language."
[b] Percent "Families with Female Head."

SOURCES: U.S. Bureau of the Census, Census of Population and Housing. Census Tracts. Final Report PCH (1). Washington, D.C.: U.S. Government Printing Office, 1972; Volume 243. Youngstown, Ohio, p. P-1, 15, 29, 43; H-1, 15; Volume 5. Albuquerque, New Mexico, p. P-1, 9, 17, 25; H-1, 9.

Tables 1 - 2 Prepared by Virginia Bronzi, Research Assistant on a Mellon Fund grant at Vassar College.

A COMPUTERIZED COURSE IN ELEMENTARY STATISTICS: EDUCATIONAL OBJECTIVES AND METHODS

Richard W. Evans
Beaver College

During the 1980-81 academic year, the Department of Psychology has been engaged in developing a computer-based course in elementary statistics under an NSF LOCI grant to the chairman, Dr. Bernard Mausner. The course has for some time been a requirement for psychology majors, and had been taught according to the Keller method of PSI instruction; that is, students advance at their own pace, receiving the bulk of instructional material from a programmed text, and take tests at the conclusion of each unit. Grades were pass/fail and students were allowed to take a given unit test as many times as they wished. Throughout, tutorial assistance from students who had previously completed the course was available.

We now have a completely computerized elementary statistics course on the PDP-11 computer maintained by Beaver College. The CAI system programming was done by Edward N. Wolff of the Mathematics Department of Beaver College, and employs a CAI language of his creation. There are 16 instructional units covering topics in elementary statistics from simple descriptive statistics to one-way analysis of variance. These units have been designed, written and implemented on the computer by the author of this paper.

The psychology department entered the area of computerized instruction in what was hoped to be the last in a long series of attempts to give our students a genuine understanding of elementary statistics. Statistics is the backbone of experimental psychology, and is required in conjunction with a two-semester experimental psychology sequence in the sophomore year.

The psychologist needs relatively simple statistical knowledge consisting of a handful of core concepts. These central concepts, when sufficiently grasped, guide decision-making about the appropriate employment and interpretation of a variety of statistical tests. Students often encounter real difficulty here: while the concepts are not particularly difficult, they do require a degree of thought to master. In addition, students need some ability to work with algebraic formulae and numbers. In combination, these two demands can produce deep anxiety. Such anxiety usually renders the student incapable of genuinely mastering statistics; the result is that he abandons any attempt to understand the material and resorts instead to rote memorization.

Beaver College encounters many psychology majors with little experience in mathematics. At the same time other major students are quite familiar with math and find elementary statistics moderately enjoyable and challenging. In an attempt to create a course which serves the entire population of psychology majors, it was decided early on to switch from a lecture format to the Keller method of personalized instruction so that students could work at their own pace. With this instructional format, students worked at their own pace from a workbook and student tutors administered tests on a pass/fail basis. The course avoided student anxiety which would have been encountered in a lecture aimed at the student of average ability. The Keller method, however, did have a drawback: students completed the course understanding almost nothing of statistics. This was quite a puzzle. Students were bored with the course; they did not find it interesting or exciting - but they did complete it, passing the unit tests. Occasionally though, one of the better students would remark that she didn't seem to be learning anything, though of course, she could not be very specific about the nature of the problem.

I began to get a sense of the thinking of the students when I attempted to revise the unit tests. The tests were essentially the same as those sent to us by the workbook's publisher. They had been used unchanged for a number of years, and it seemed they might be a source of difficulty - - they did not seem to put any emphasis on understanding the concepts in the unit. Each test item simply tested a different computational procedure.

For example, an item might present two groups of data such as IQ scores from students in a classroom, and it would then tell you to calculate the standard deviation (which is a measure of the degree to which items in the sample vary from each other) of both samples. This is a very unsatisfactory test item in my opinion. It does not indicate whether the student understands what the standard deviation represents; he simply knows how to calculate it. The tests were not sufficiently thought provoking to lead to a real understanding of statistics. The next step, then, was to rearrange the tests.

Perhaps, we hoped, rewriting the tests would lead students to look for the appropriate information while they worked through the workbooks and thus perform better on items which tested their understanding of the material. The new test item might read: "Here are two samples of data: which is the more variable? Explain your answer." The answers to this type of question were surprising; most came in the form of blank statements. There were some cries of anger and a few expressions of extreme frustration and hysteria. Occasionally we did get answers. Someone might have the presence of mind to realize the chapter actually had something to do with variance, and would then write something which he had clearly spent considerable time memorizing, like, "variability is the extent to which the members of a group differ from each other." That would be all. Not knowing what to do with this definition, the student would let it stand as his answer to the question.

Occasionally someone would read that definition over a few times once they had it on paper and realize they could use the definition to compare the two groups. The student would count the number of occurrences at one value, and those at another, and finally write a laborious paragraph comparing the variability of the two groups by contrasting the number of items at each value. The students almost never recognized that the easiest way to compare variability would be to calculate the standard deviation for the two groups and compare them.

This finding is something of a shock. Here's a student who is obviously quite bright since he or she is on the threshold of inventing a measure of variability, but who cannot recognize that the problem's solution lies in the numerical measure which had been covered for many pages in his workbook unit. The students, knowing full well how to calculate a standard deviation, had no idea what this numerical measure represented.

The new tests did not help. The students simply could not do them. The major problem, of course, was in the workbooks the students were using. In an area where learning the core concepts was essential, the workbooks were promoting a cognitive strategy for learning not at all conducive to understanding of the material. They contained information which allowed the student only the option of rote memorization of the material in each instructional unit. In this particular example, the workbooks simply did not present the standard deviation meaningfully. A few lines of text simply defined variability and standard deviation, but made no attempt to get the student to understand how standard deviation provided a reasonable numerical representation of variability.

It was necessary, then, to provide sufficient conceptual information. This was attempted in two ways. First, I required a simple paperback which emphasized conceptual rather than computational aspects of statistics. Second, I wrote a study guide which tried to explain the relation of the major concepts of statistics to their computational procedures. Students now had good material for the meaning of the core statistical ideas.

How then, did performance on the new test items change? Essentially there was no change. In this case the problem was a puzzling one. While I stressed the importance of the conceptual material for passing the tests, students seemed to have a severe problem integrating the material from the paperback and study guide with the material from the workbooks. Essentially, the strategy demanded by the workbooks required that students memorize computational procedures in such a disjointed way that an understanding of their conceptual underpinnings was precluded. As a result, the material in the paperback and study guide, even when it was diligently studied, could not be integrated with the computational memorizing students adopted for the workbooks. Evidently I had to write the conceptually based material in such a way that the computational aspects of statistics would be an integral part of the student's understanding of the material, rather than a separate component.

PSI VS. LECTURE PRESENTATION: THE NEED TO CONTINUALLY ASSESS CONCEPTUAL PROGRESS

A little reflection revealed the problem from the standpoint of someone putting together a workbook: it is much easier to introduce memorizable computational procedures rather than stress a conceptual understanding. The difficulty of a conceptual approach is due to the requirement that some assessment be made to determine whether the student is prepared for each successive idea.

This is what the successful lecturer (at least in small classes) does. If the lecturer is stressing a conceptual approach to certain material, students must understand one point before advancing to the next, or else they will not be able to properly comprehend the succeeding point. If the lecturer does advance too quickly, students will become frustrated. It is very difficult to check on individual student comprehension in a lecture class, and it takes great skill as a teacher to do this effectively. Most of us occasionally are guilty of giving up on comprehension and simply requiring students to memorize a point. The more heterogeneous the class is in student ability, the more difficult it becomes to keep track of the individual student's conceptual progress. This is precisely the state of affairs which drove us away from the lecture format for our statistics course. In large lecture courses this problem becomes particularly acute to the point where minor objective points are stressed, and progress is tested by multiple choice exams.

This was the problem faced by our workbooks. Since there was no possibility of adequately assessing individual conceptual progress, the workbook stressed instead a multitude of computational procedures which could each be presented in a manner largely independent of the remaining computational procedures. Rather than deal with comprehension at all then, it dealt simply with computational procedures, avoiding problems that stem from differences in the manner and rate at which students comprehend material. Instead, the workbooks presented material which would not differ in form from student to student, and which necessarily resulted in statistical competence of a very mundane sort.

EDUCATIONAL ADVANTAGES OF THE COMPUTER

Diagnostic Procedures

My goal was to stress conceptual skills and present computational procedures as derivative of the major statistical concepts. The consequent need to assess individual conceptual progress led to the selection of the computer as the primary instructional medium. The major requirement, then, was to program instructional material which was individualized in the sense that it would be sensitive to the individual's comprehension, rather than to the individual's rate of memorization of computational formulae. This stress on conceptual knowledge and on assessment of the student's conceptual progress required that the computerized-instructional material contain routines designed to determine the nature of the student's understanding of the concepts in question, and adjust its presentation to guide that particular student to complete understanding of the material.

A great deal of work has been done recently to develop the means of assessing individual knowledge of a particular area (Resnick, 1976; Greeno, 1980). Work by Brown (Brown and Burton, 1978) to develop cognitive diagnostic techniques for computer implementation has been especially well received. Implementing such diagnostics in instructional programming requires a considerable amount of cognitive research as well as instructional programming. In general, this type of programming is very labor intensive, and I found there was time to provide a comprehensive assessment only in a few units. To a great extent, then, I continued to use CAI tutorial presentation and drill and practice, stressing as much as possible an understanding of the concepts involved.

Problem-Oriented Instruction

For those instructional sequences which did contain very intensive, diagnostic routines, a consistent pattern of instructional design gradually emerged as most appropriate. I call this pattern of instructional design "problem-oriented instruction." In the case of elementary statistics, problem-oriented instruction consisted first of presenting the student with data which the student was gradually aided in analyzing. This required the student to act as a problem-solver, rather than a passive learner. The diagnostic routines were used to assess his understanding of the concepts relevant to the problem and to guide the student to a successful solution. This approach was considered particularly desirable since students would actually discover how a statistic provided a solution to a particular conceptual problem. Students would thus develop very useful knowledge. They would have an understanding of statistics so they could decide what problems they were concerned with, and they would know how to get from these problems to some appropriate computational solution. These problem-oriented instructional sequences were especially promising. The student's computational knowledge came to be embedded in a conceptual context. Since that conceptual context corresponded to the student's comprehension of a statistical problem, that problem would, when comprehended, access the computational procedures which led to its solution.

An early example of this type of instructional programming was performed with data the student generated. The student was asked by the computer to engage in an experiment with himself as subject. The computer then required the student to memorize 15-item lists of words. After each list, the student recalled as many words as he could, reporting them to the computer which then evaluated his performance. The student was then asked a number of questions about the data and what they said about his memory abilities. Finally, the student was given the results of others to compare to his own. At first, the results of other students were readily comparable since they had each performed on an equal number of lists, namely, five. Eventually, however, the computer introduced a student for whom results had been collected on eight different lists. These results were selected so that the total of the eight lists would exceed the first student's total, but the average per list of the second student would be lower than that of the first student. The first student needed to resolve this problem: while he believed that his own memory performance was better than that of the second student, the second student had generated a greater number of correct words. The solution of course is to take the mean or average number of correct words per list, and the computer then gradually worked the student through the steps which lead to that realization. This problem-oriented format of instruction, which stressed the function and necessity of the ideas in each area and introduced the computational procedures as ways of expressing an idea which is itself the solution to a problem, seemed quite effective in those cases where I was able to spend the time required to achieve it. In the future, as the course is improved more of these problem-oriented instruction sequences will be designed and programmed.

EVALUATION

Part of the project has been an evaluation of the effectiveness of the program. This evaluation is still underway, and when completed, it should be clear whether students learned as much or more of statistics with the computer than with the workbook-based course. In addition, students' evaluative reactions to the course have been collected. At the conclusion of each unit, students were asked to fill out an evaluation form, and there are some preliminary results. In analyzing the responses to this questionnaire, the following things become clear: students find the computer units as valuable as the workbooks for interest and ease of comprehension. At a highly significant rate, students prefer the computer units over a lecture as a method of presentation both for the degree of interest and the amount of material they gain from a given unit. In view of student preference ratings, then, these preliminary data strongly suggest that the computerized version of the course is successful.

To some extent the stress on conceptual matter has also proven successful. The most positive sign to emerge is the reaction of students to the midterm and final exams. These two exams were added to the unit tests employed during the previous years. The unit tests were retained in much the same form as when the workbook was the sole source of information for the course. So, the unit tests primarily tested the computational skills stressed by the workbooks. Students found the unit tests to be undemanding of the conceptual material. The midterm and final, however, were of a quite different nature. They each presented the results of an experiment and then required the student to compare the two groups of data and make conclusions in ways relevant to the experimenter's hypotheses. The response to the midterm bordered on panic. The students were required to assess the experimental situation, determine what questions should reasonably be asked, and then select and follow the appropriate computational procedures. One-half of the class had great difficulty making the transition from the unit test approach to this problem-solving format. For approximately one-fourth of the class, the task provoked anxiety and these students either failed the test or could not complete it on the first attempt. Following the difficulties encountered on the midterm, however, these students showed impressive abilities to adjust their style of learning and absorb the more conceptually-oriented information presented in the computer units. The result was a marked increase in the degree of success on the final exam, even though the conceptual demands of the final were considerably greater than those of the midterm. In general, the students found it possible, once the midterm impressed on them the need to engage in conceptual processing of the CAI material, to do this processing, and become less dependent on the computational memorization stressed by the workbooks.

There has been some success, then, in the program. While it has not been possible to build in a high density of problem-oriented instructional sequences with the requisite diagnostic procedures,those sequences which have been created seem very effective.

By employing these problem-oriented instructional sequences and by stressing conceptual understanding of the material in the tutorial and drill and practice modes, the students acquire a genuine understanding of the material. The difficulties encountered by some students at the midterm demonstrated that the system is at present by no means perfect. In an attempt to improve the program the existing units are being analyzed with the intention of designing problem-oriented instructional sequences for all of the major conceptual points covered by the course. Ultimately we hope to have an instructional package which will make proficient statistical problem-solvers of our psychology majors.

REFERENCES

Brown, J.S. and Burton, R.R. Diagnostic models for procedural bugs in basic mathematical skills. Cognitive Science, 2, 155-192 (1978).

Greeno, J.G., Trends in the theory of knowledge for problem solving. In Tuma, D. and Reif, F. (eds), Problem Solving and Education, 1980, Hillsdale, N.J.: Lawrence Erlbaum Associates.

Resnick, L., Task analysis in instructional design: Some cases from mathematics. In Klahr, D. (ed). Cognition and Instruction, 1976, Hillsdale, N.J.: Lawrence Erlbaum Associates.

GAMES PEOPLE WILL PLAY: DEVELOPMENT OF A METHOD TO ASSESS INTEREST IN INSTRUCTIONAL GAMES

Jerry D. Neideffer and Selby H. Evans
Department of Psychology
Texas Christian University
Fort Worth, Texas 76129
(817) 921-7410

"Man is a gaming animal" according to the essayist Charles Lamb. He must have been correct, given the fact that one large-scale use of integrated circuit technology is electronic games. In 1979 the demand for electronic games was so great that the supply of chips was inadequate, (Wiswell, 1980). Milton Bradley's sales were led in 1978 and 1979 by a light and sound imitation game called Simon (Morris, 1980). A rash of copies followed along with new and more original entries, leading to the current total of some 400 electronic games on the market. One game, Space Invaders, reportedly is the most popular game in arcade history, (Morris, 1980; Wiswell, 1980). Star Trek-based games are also very popular.

Today's advanced technology in electronics and psychology opens the door for consideration of combining the practicality and sophistication of integrated circuit electronics with the highly motivating nature of electronic game playing, and use of these games to teach. Moss (1980) reports that 58% of computer center directors surveyed believe electronic teaching methods will be used as an alternative to the traditional classroom for at least 25% of the total curriculum, and 72% indicated they thought students would have personal computers available for homework assignments by the year 2000. These figures suggest there should be no limit to the types of games which could be made available, if not in portable form as part of the standard homework. (How I would have loved to have heard my elementary school teachers say, "Your home-game assignment is....")

Using games to teach is not a new idea. The Montessori (1912) method of teaching relies heavily on activity and games. Dewey (1915) also includes games in his conception of education. He notes that all peoples have used play and games for much of the education of children, especially the young. Dewey's perception that play is important in education led him to assert that few educational writers acknowledged in theory the status held by play in practice. The concept of learning through game playing is well grounded in tradition, but a systematic means of engineering instructional games is still lacking. Developers of a technology of instructional games must be prepared to address a number of issues if games are to be effectively used in formal education.

There must, of course, be general strategies for developing an instructional game to meet specified behavioral objectives. Developers must agree upon concrete behavioral terms which will be used to specify these objectives and an ability to ascertain when they have been met. Criteria for deciding what type of games will best fulfill the behavioral objectives must be established as well. The game designer needs to decide how best to couch the instructional material within the framework of the chosen game. Procedures for making these decisions must be developed and validated.

The presentation rate and level of difficulty of the instructional material should be such that bright students do not become bored nor slow learners frustrated by failure. An intermediate rate and difficulty will be required if the player has no choice of proficiency level; otherwise these parameters may be selected by the player who may over or under estimate his ability. In either case, a means of accurately assessing the player's progress must be devised so that rate and difficulty can be adjusted constantly during play. The accuracy of this assessment must also be validated.

Time requirements for both student and teacher are an important consideration in any aspect of education. Comparisons will be necessary to assure the instructor's time is at least as well spent, if not more efficiently used, when games are employed rather than other methods. Criteria for comparison must be established because games and other methods cannot be simply compared on time required to learn alone. Evans (1978) points out that what is normally considered "recreation time" may be spent playing instructional games, such as the statistics game, Statrek, which he developed, (Evans, 1978). Statrek has proven to be immensely popular, not only with students who can play for class credit, but also among graduate and undergraduate students who simply play for fun. Thus, even though more time might actually be needed to present a specified amount of instructional material, a game might be more efficient if less class and home study time are required.

Games must be compared to alternate forms of instruction in a more general way; not all material may be equally suitable for presentation in a game format. Procedures for deciding what elements of instructional material indicate or contraindicate the use of games must be an integral part of a technology of instructional games. A related problem is the necessity for procedures that will identify characteristics of individuals, which suggest the probable success or failure of teaching via games on an individual basis. Effectiveness of instructional games for both types of material and types of players must be assessed in terms of the specified behavioral objectives.

Before any teaching method can be accurately evaluated one must first be assured that the method has been accurately represented. For example, the amount of material initially learned or retained from a lecture can reasonably be assumed to be, in part, a function of the instructor. An uninteresting speaker simply does not hold the attention of students. Any effectiveness by the lecture method would be tainted by an unmotivating presentation. Likewise, before an instructional game can be assessed as a teaching tool, one must first be certain the game has been well designed as a game.

No game, no matter how well designed in its educational aspects, can teach anything if it is not played. To be an effective teaching tool a game must be interesting. What makes them interesting is currently a matter of speculation. McVay (1980) believes that a high amount of player interaction, maximum choices for the player, simplicity, and the opportunity for creativity are important aspects. Wasserman and Stryker (1980) have touted multimachine games (more than one computer) as being more interesting than single machine games because they alone have the following characteristics: More than one human player is involved; success depends on careful use of incomplete information; and play occurs in real time.

To empirically determine what makes games interesting to play, one must make several determinations. First, what are the relevant dimensions of interest as the term relates to instructional games? What elements of games lead them to be judged by game players as uninteresting? These dimensions may or may not be the same, i.e., uninteresting may not simply be the opposite of interesting. If so, one should know both sets of dimensions when designing an instructional game so that interesting elements can be included while uninteresting elements are avoided.

A procedure to measure the degree to which a game has or lacks elements which make it interesting can serve a two-fold purpose. First, the instrument could be used to predict player acceptance of a new game. Perhaps even more useful, however, would be to identify the specific dimension of interest along which the game is deficient. This knowledge would allow improvement of both new and existing games.

Difference among game players may lead to different conceptions of interesting and uninteresting. It would not be surprising, for example, to find two groups of game players; that is, one liking fast paced, highly competitive games, while the second enjoyed more leisurely, relaxing games. These differences may lead to formulating more than one set of dimensions for interesting and uninteresting, but will more likely suggest the need for flexibility so that any given game can be made interesting to the largest percentage of players possible. How differences among game players should be treated by designers must be determined.

We set out to meet two objectives with the present study. We wanted first to collect and test a preliminary set of elements to define the dimensions of interest in games. Once these were known, games could be located in a descriptive space delimited by them. Games not sufficiently interesting could then be identified, suggesting possible ways to remedy this problem. The second objective was to examine the response patterns of the participants to see if subgroups of players exist for whom games might have to be modified.

Open-ended interviews about what makes games interesting were conducted with a group of undergraduate student game players. Participants were asked to talk about what elements make games interesting to play, what makes them uninteresting, and what they would include in a game of their own design to assure it would be interesting. Twenty-one descriptive terms and phrases were collected (see Table 1).

Subsequent groups were asked to rate games which they found interesting and uninteresting on a seven-point Likert scale for each of these 21 elements. For example:

Competitive 1 2 3 4 5 6 7

Participants were instructed to circle the number representing the degree to which the game possessed each element. A 1 indicated that the game possessed the element to a very low degree while a 7 indicated the game possessed the element to a very high degree. Each participant named and rated one game he considered to be interesting and one considered to be uninteresting.

Backgammon and Scrabble were the most commonly selected games. About 20 board, card, and electronic games composed the entire group which were rated. We hope this fairly broad range will lend generality to our method in terms of the presentation media to which it will apply.

Data were analysed on a Xerox Sigma-9 computer using an APL based interactive system called Protables, designed by Evans (see Evans and Gage, 1979; Evans, Gage, and Neideffer, 1980; Evans, Neideffer, and Gage, 1980). Analyses were conducted from an exploratory rather than confirmatory perspective. Multivariate techniques, plots, and histograms were the primary tools.

Rating the games on 21 items allowed us to represent their interestingness in a 21-dimensional space, a conceptually difficult problem. We sought to reduce the number of dimensions via principal components analysis (PCAN). PCAN effectively reduces the number of dimensions while still accounting for most of the variance in the original set of variables (Evans and Attaya, 1978).

When game ratings were analysed via PCAN, three principal components (PC) were observed (see Table 1). Each PC is a new element of interest in games formed from the original 21 items. Values shown in the table are correlations between each original item and the three PC's. The numerical value for any game on any PC is a weighted linear combination of ratings on the original items. The first PC is best represented by items identified as "challenging," "requires skill," and "testing capabilities to their limits." The second PC is related to the length and pace of the game and to the role of chance factors in determining the game's outcome. The third PC is best represented by "cooperation." The stability of these factors remains to be seen in replication with more carefully controlled data collection procedures.

Q-type factor analysis, a form of factor analysis which reduces the number of participants to some minimum number of clusters, was then conducted. If the sample of participants is homogeneous, i.e., most came from the same population, one factor would account for most of the variance among participants. This was not the case for our game players. Two factors were required to account for about 50% of the variance. It is therefore reasonable to think of the sample of game players as consisting of players of two distinct types.

When these subgroups were compared element by element, we found that they differ in their characterizations of "interest." One group indicated a preference for games that are relaxing, not too long, fast paced, and have outcomes largely governed by chance. Creativity and learning something from the game are not important elements for this group. The second group prefers games which test their limits, are fast paced, require creativity, and teach something during play. They want little left to chance and are not concerned with relaxation. These differences are plausible because they define two likely groups of game players: those who play to relax or as a diversion, and those who play for tough competition.

Because people who appeared to come from a single population disagreed about what makes an interesting game we believe this information will be important for further study. Either separate games will have to be designed for people who find different aspects of game playing interesting, or games must be constructed that suit the preferences of the players.

Our findings are preliminary but encouraging in that our methods of investigation appear to be sound. We consider this study only the digging of the foundation, but we are now preparing to pour the cement. Shortly, we plan to have participants play and rate a limited number of games in a controlled environment. All participants will play the same games under the same conditions. We also plan to identify physical attributes of the games which might help to predict level of interest the games will generate. These might include game length, time between plays, number of colors in the display, or more complex measures, such as amount of motion. With these ratings we hope to produce dimensions of interest to indicate how interesting games are by their relative positions on these dimensions. Finally, we hope to demonstrate a change in level of interest by manipulating game elements which result in movement of a game along some dimension(s) of interest.

We then plan to re-examine our data as suggested by the appearance of subgroups on the QFAC analysis. When this project is completed we will have devised a method to study interest in instructional games. Although our goal is to assist designers of instructional games, we would of course be pleased if our method proved to be useful for instructional design in a more general sense.

TABLE 1

Correlations Between Items and Major Components of Principal Components Analysis

	P1	P2	P3	Proportion of variance
Challenging	.83		-.28	.76
Requires Skill	.77	-.29	-.24	.73
Tests limits	.75	-.32		.68
Problem solving	.71	-.24		.57
Fast thinking	.68			.50
Tough opponents	.68			.50
Strategy	.66	-.29	-.32	.62
Competitive	.66		-.45	.64
Varied versions	.63		.31	.50
Teaches something	.62	-.31	.35	.61
Alternative moves	.61			.41
Diversion	.61	.36		.56
Creativity	.56	-.32	.46	.62
Game length	.54	.59		.66
Fast paced	.54	.54		.58
Specific goal	.50	.25	-.31	.41
Cooperation	.39		.57	.49
Rules clear	.33	.44	.28	.38
Waiting time	.33	.55		.41
Relaxing	.33	.38		.25
Luck		.67		.46
Proportion of VAR	.35	.12	.07	

REFERENCES

Dewey, J. Schools of To-Morrow. New York: E. P. Dutton and Company, 1915.

Evans, S. and Gage, F. H. APL programs for interactive data analysis: Basic statistics and histograms. Behavior Research Methods and Instrumentation, 11(6), 1979, 605-606.

Evans, S., Gage, F. H., & Neideffer, J. D. APL programs for interactive data analysis: Data entry and correlation. Behavior Research Methods and Instrumentation, 1980, 12(3), 372-375.

Evans, S. How the Klingon Wars can teach basic principles of elementary statistics. Unpublished manuscript, 1978.

Evans, S., Neideffer, J. D., & Gage, F. H. APL functions for interactive data analysis: Graphics and labels. Behavior Research Methods and Instrumentation, 12(5), 541-545.

McVay, P. O. Tapping the appeal of games in instruction. Proceedings of the National Educational Computing Conference, 1980, 271-275.

Montessori, M. The Montessori Method. Cambridge: Robert Bentley, Inc., 1973.

Morris, S. The ten best games of the year. Omni, 3(3), 1980, 104-106, 135-137.

Moss, J. L. Directions of computing in higher education: Predictions from university computer center directors; The results of a survey. SIGUCC Newsletter, 3, 1980, 5-10.

Wasserman, K. and Stryker, T. Multimachine games. Byte, 5(12), 1980, 24-40.

Wiswell, P. The name of the game is electronics. Parade, November 23, 1980, 12-15.

COMPUTERS AND THE NURSERY SCHOOL

Dr. Kathleen M. Swigger and Dr. James Cambell
North Texas State University

You wander down the hall, through the open door, and look around the room. There sets a microcomputer surrounded by small children who are busily pressing keys, giggling, and watching the screen.

At first glance, there seems to be nothing particularly unusual about this scene. It is being reenacted daily in school districts across the country. The only difference is that the children in this case are not high school students or even grade school children, but rather three and four year olds who can barely sit still much less interact with a computer. Children from the North Texas State University Nursery School were introduced to a computer this year through a series of programs developed for the TI 99/4 microcomputer. During the 1980-81 school year, we worked with the nursery school and the children to develop computer activities that could be incorporated into their respective programs. This paper describes both the programs and experiences that resulted from the introduction of the computer into the preschool environment.

PURPOSE OF THE PROGRAMS

The technology of microelectonics has already had a profound impact on secondary and elementary school children. We now know that computers can be used by students to solve problems, to learn basic skills, and to shoot down alien space ships. Children across the country are learning to manipulate sophisticated machinery to derive answers to problems and to have fun. But these kinds of activities have not had much effect on small children, particularly three- and four-year-old children. Questions still remain as to whether this audience is capable of using a computer. Is it necessary, for example, to know how to read before you can use a computer? Is it possible to teach small children how to control the keys, etc.?

The above questions and more were asked in an attempt to discover the feasiblity of exposing preschool children to the world of computers. We assumed that a child learns almost everything from his environment,and that a computer could be introduced as part of this environment to accelerate acquiring certain learning skills. We maintained that children could learn, through computer-based play, that they are not impotent, that they can solve problems, manipulate machines, master skills, and be powerful. All of these concepts are important to children, regardless of their age. However, three- and four-year-old children are especially vulnerable and need to believe that they have some control over their lives. Therefore, we were trying to discover whether it was possible for small children to gain control over a computer. And, just as important, we wanted to know what would happen to the children once they had this control. In addition, we wanted to assess the impact of the computer upon gender preferences and competencies.

COMPUTERS AS LEARNING AIDS

When one mentions computers in education, a concept of programmed instruction delivered step-by-step through the computer comes to mind.

This is indeed the classic definition of computer-assisted instruction (CAI) and is one used to persuade young students they are engaged in a "Platonic" dialogue. CAI uses a teaching/learning model based on a shaping process: the teacher (or experimenter) reinforces the student for making successful approximations of the teacher's objectives. A careful task analysis of the desired performance is used to design the sequence of instructional frames. [1]

This model for CAI has been successfully implemented in high schools and colleges for a number of years. Tutorial programs incorporated into the home or classroom have permitted children to master skills (such as mathematics and speaking a foreign language) whenever they have chosen to do so, rather than as part of a group progressing toward the same goal. Large CAI systems such as CDC's Plato have been implemented and are being used in learning centers throughout the country. Similarly, math drills and programs designed to teach reading skills have or are being developed for Pets, TI's, Apples, and TRS- 80's. We can safely conclude that computers and computer-assisted instruction have become an integral part of our children's educational experiences. However, these learning experiences have not always been directly translated to the younger set.

Three- and four-year-old children have been ignored by computer specialists because of the obvious limitations of input/output devices. If you are forced to rely on a CRT and keyboard as an I/O device, then you are limited to teaching only those who can read or, at the very least, recognize the alphabet. Even Texas Instruments' Speak & Spell and their new Speak & Read requires that the child recognize letters and numbers.

Personal computers are now being developed that place more emphasis on user accessibility and multiple I/O modes. Spoken words, a touch display screen, and a musical keyboard, all provide computer input that appeals to a small child. Robert Taylor's group at the Teachers College at Columbia University is working with kindergarten children in the New York School district to determine the possibility of using computers with this age group. Seymour Papert, speaking on the first day of the International Federation for Information Processing (IFIP's) Congress described an experiment in Dallas where a nursery school teacher has written a number of programs that enable three and four year olds to manipulate brightly colored objects on a computer screen by hitting a small number of keys marked with arrows to indicate directions of movement and colors to indicate color change. [2] We are now learning that interactive electronic systems can respond or give directions with letters and numbers, and they can also tell the child to perform a task by playing back a recording, synthesizing speech, producing music, or generating an image. These powerful new additions to the computer enable us to "talk" to the preschoolers and to instruct them to perform certain tasks.

DESCRIPTION OF THE PROGRAMS

In our work with preschool children at the North Texas State University Nursery School, we have seen that a computer can assist the initial learning of basic skills and sophisticated concepts. We have seen children sit down in front of a computer, use the programs, and complete the required tasks. Rather than prescribe the child's learning experience, the computer presents an inviting toy with which the child can interact, freeing him or her from an adult's limited abilities to provide optimal educational experiences.

The specific learning materials designed for preschoolers at North Texas are aimed at increasing a child's reading readiness skills. The computer programs include having the child select the shape that is different, match various shapes, discriminate between two items, estimate various quantities, and make comparisons. The children make their responses by moving a "line" across the screen and then pressing a specially marked key when the line is in the correct position. (See Figure 1) A response

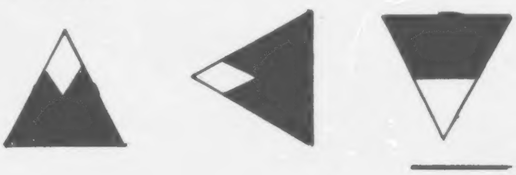


Figure 1: Example of Student/Computer Interaction

is given only when the child selects the correct answer. Our preliminary tests indicate that a very pronounced system for incorrect responses, such as a voice that says "Uh Oh", frightens some children. Our program merely sounds a short "bleep" whenever the child makes an incorrect response. Each of the programs is approximately twenty minutes long and can be used by one child or an entire group of children.

SOME PRELIMINARY RESULTS

Although we have just begun to examine the possibilities of using computers with preschool children, we have already made some very positive discoveries. Most importantly, three- and four-year-old children love computers. The power of interactive systems to attract children, to invite them to "play," and to give them answers, approximates the learning process that occurs when a child learns to talk, walk, play an instrument, etc. Given minimal directions, all the children were able to master the skill of manipulating the figures and pressing the correct keys. None of the children appeared afraid of the machine or incapable of understanding any of the programs.

We also observed that the children appeared less anxious about the machine and less apprehensive about the materials when adults were not physically present. The film we shot during the demonstration shows a distinct difference in the children's attitudes whenever the adults leave the area. It is at this time that the children relax and truely enjoy the play experience that comes from using the computer. They become fascinated with the keys and with the speed of the computer's responses. Not surprisingly, a three- or four-year-old child responds much like an adult when left alone with an intriguing toy.

Another result of the experiment occurred in the actual mastery of specific skills set forth in the programs. During the testing period, several teachers would comment on specific children, stating this or that child was not quite ready to master a particular skill. Invariably the child would sit down in front of the computer, listen attentively to the instructions, and perform the required task successfully. As educators, we must be aware of restricting educational opportunities for our students or narrowly defining the parameters of learning that a child is capable of. This condition impedes learning, eventually creating learning problems that beset almost all children in school. On the other hand, with a feeling of self-confidence that comes from using the computer, children can probably learn more complex tasks than we think them capable of learning.

INTERACTIVE GAMES AND TINY TOTS

One area that holds great promise for very small children is that of computer games. If one builds into games an opportunity to increase skills systematically (such as doing something faster each time) or to advance to a more complex or difficult task, then a computer game can provide a very valuable experience. Hand coordination can be vastly improved by forcing the child to press the correct keys. Eye coordination is also greatly enhanced. Justification for these games is simply that they can be used to improve a child's coordination, which is extremely important for the acquisition of basic skills such as writing and reading skills. Computer games can also simultaneously incorporate fantasy elements, systematic responses, and competition as well as foster teamwork, cooperation, and cross-age helping.

Of course, interactive games and instruction can get boring, like any new toy. Variety and increasing challenge must be built into computer-based activities, or they may become tedious. Apparently this problem does not arise as often when working with computer-based materials as it does when working with other types of learning materials. Most of the N.T. Nursery School children, for example, were playing with the computer thirty or forty minutes after being introduced to the machine. Of course, some of this may disolve in the future, but for now, the children's attention spans have been expanded.

SUMMARY AND FUTURE ACTIVITIES

What we have discovered is simply that small children can and will use a computer. The questions that now remain are what activities can best be presented via computer, what techiques can be used to present materials, and, finally, what effects computers have on very small children. These questions are extremely important for people who are concerned with computers in education, and the answers hold much importance for future generations.

First, what sorts of tasks can best be presented to small children via computer? That is, which skills or tasks need to be presented without teacher influence or dominance. Several educators have conjectured that small children do certain things because it pleases adults rather than the children themselves. In the brief film of the North Texas children, we noticed that whenever adults were present, the children responded to the computer and then looked immediately toward the adults for approval rather than toward the computer. When the adults left the area, the children engaged in a play-like activity with the computer and peformed the required tasks faster.

What types of techniques should be used in the presentation of good quality computer-based learning materials? Are certain colors more attractive to small children than to older children? We discovered that many of the computer-learning materials for small children are filled with unnecessary graphics that tend to confuse the child rather than explain the task. We also found that speech can sometimes be detrimental to good computer-aided instruction. As previously mentioned, the simple utterance "Uh Oh" after an incorrect response elicited very negative feelings among the children. Therefore, one must be very careful about the intent and purpose of the educational materials.

Third, what effects can we expect from the computer, and how will it change our children's personalities? There is still debate over how the television has changed the psyche of our youth, and how it has changed their learning styles and cognitive development. Will the same be true of the computer? The N.T. Nursery School children thought the computer was just a T.V. As a result, the children had little difficulty adjusting to it.

Finally, can the computer eliminate the sex differences that are sometimes associated with certain learning activities or environments? Recent studies have confirmed the increasing data regarding gender differences in aptitudes and abilities. If this is the case, perhaps computers can be used to eliminate some of these differences. If the computer, like the T.V., can be perceived by children as sexless, then perhaps this inanimate object can be used to teach those concepts or skills that require objectivity. Can a boy learn verbal skills better via computer than he can from a teacher? Can a girl gain self-confidence in mathematics by using computers? The answers to these and many other questions can greatly effect the learning ability of future generations. With help, we hope to perform several studies that will clarify these and other issues. We intend to film the chilrens' formal and incidental experiences with the computer to better understand the nature and effect of the interactions on preschoolers.

We are extremely optimistic about what the computer can represent to children of the future. It is estimated that by the year 1985, 80% of the women in this country will be working. As a result, a substantial number of our children will be cared for in nursery schools and day-care centers. Many of these preschools and day-care centers are understaffed, poorly equipped, and provide very little in the way of a stimulating environment. A computer could eliminate some of the gross inadequacies that presently exist in such places. It could, in short, be a Sesame Street that talks back. If the computer can provide some relief in just this small area, then it would indeed have a very positive effect on our very young children.

References

1. Banet, Bernard. "Computers and Early Learning," Creative Computing, Sept. -Oct. 1978, Vol. 4, No. 5, 90-94.
2. French, Nancy. "Computer Seen as Pencil in Child's Learning," Computerworld, October 13, 1980, p. 36.
3. Kay, A., & Goldberg, A. "Personal Dynamic Media," Computer, 1977, Vol. 10, No. 3, 31-40.
4. Moore, O.K. "The preschool child Learns to Read and Write in the autotelic Responsive Environment," in Y. Brackbill & G.G. Thompson (Eds.) Behavior in Infancy and Early Childhood. New York: The Free Press, 1967.
5. Papert, Seymour. "Teaching Children Thinking," Cambridge, Mass.: MIT Artificial Intelligence Laboratory, LOGO memo No. 2 AI Memo No. 247, 1971.

COMPUTER-ASSISTED INSTRUCTION IN READING USED FOR KINDERGARTNERS AND FIRST GRADERS IN DILLON, MONTANA, 1979-80

Nellie Bandelier
First Grade Teacher
Mary Innes School
Dillon, Montana 59725
406-683-5347

Computer-assisted instruction for teaching mathematics and physical sciences has been used in secondary schools for over ten years; however, computer-assisted instruction in other disciplines and at the elementary level has lagged behind. With the development of microcomputers and the reduced cost of hardware, CAI is now within the budget of elementary schools, but there is an absence of suitable software. With this in mind, an effort was made by the author to adapt CAI to teaching kindergarten and first grade students reading at the Mary Innes School in Dillon, Montana.

A one-year incentive grant of approximately $2,900 was obtained through the Office of Public Instruction in Helena, Montana, to develop this program. The grant was used to purchase a microcomputer and to reimburse a teacher and secretary for their services. The programs were developed in the summer of 1979 and used by eight classes, four kindergarten and four first grade during the 1979-80 academic year.

SELECTION OF THE MICROCOMPUTER

The Commodore Pet 2001 was chosen for a variety of reasons, but the major factors were convenience and simplicity: the teacher has only one piece of equipment to move into her room on a cart and one plug to insert into the electrical outlet. Loading programs on the cassette, which is built into the computer, is also easy.

Another desirable feature of the Pet is that both capital and lowercase letters, which are emphasized in reading skills, are available; many do not have lowercase. However, the Pet cannot show both lowercase letters and pictures on the screen at the same time, so rebus sentences used in the project are capitalized.

The graphics possible on the Pet add interest for the students. Though animation and pictures supply variety to the programs, graphics are time consuming and animations are difficult for a beginner to program.

Finally, the cost of the Pet was within the budget. A printer is not used although it shows which words and questions a student misses; however, it added cost and complications to the project.

PROGRAMS

Thirty programs, providing CAI of the vocabulary used in kindergarten and first grade, were prepared. These programs supplemented the reading program already in use; that is, reading skills were taught by the teacher, and then the microcomputer was available for students to practice that vocabulary.

The simplest program involves matching capitals and lower case letters: the screen shows a lowercase letter, example "a," on the screen, and the student looks for the "A" on the keyboard. If the correct key is pushed, Bobby, an animated robot, appears on the screen and tells the student the answer is correct, and the next letter is shown. The first program features the first twelve letters that children study in their reading readiness books. Another program features the next twelve letters and another has all the letters of the alphabet to match. These alphabet programs are normally used in late kindergarten and for review in early first grade. However, some kindergartners started in the fall because they were already familiar with the letters. Other programs feature rebus sentences using fifteen words taught in kindergarten, with one word left out. Example:

He is ____________ in the

1. the   2. not   3. in

The child is to press the number of the correct word for the sentence.

As the child's reading vocabulary increases, the complexity of the programs increases. In easier programs, only the missing word needs to be chosen. Next, words the child has in the reading classes are introduced in sentences without the pictures, then in short paragraphs, and then in stories. Questions are then asked about a short paragraph just read, but still appearing on the screen to check comprehension. When a story is read, the child needs to remember it because the story disappears before comprehension questions are asked. If the wrong answer is given, the story may be read again.

Two games appearing in computer magazines were adapted using selected vocabulary from the readers. A teacher's guide includes a brief description of each program and tells which book's vocabulary is emphasized to enable solving of programs that review vocabulary most needed. This part of the teacher's guide is included in the appendix.

Since the tape recorder in the Pet takes so long to find a program on a cassette, only one program was stored on each tape. This simplified and shortened the time a teacher needed to load a program in the Pet; in a primary classroom, teacher-time is at a premium.

PROGRAM CONSTRUCTION

The construction of each program varied, but generally involved two major steps. First, the sentences, paragraphs, or stories that the student was to see on the screen were written; then the material was programmed. After these two steps, the program was stored on a cassette.

An example is given here to illustrate how one of the simpler programs was done. Tigers, the first primer used in first grade, has the following vocabulary:

a
and
are
can
cat
come
fish
get
go
have
he
help
here
I
in
is
it
me
not
on
real
see
stop
the
this
to
want
we
where
will
with
you

Sentences were composed using only this vocabulary. Each time a word was used, it was marked to avoid unnecessary duplication, allowing as many words as possible to be introduced. Sentences using this vocabulary include:

A. I _________ go in there.

1. can 2. in 3. the 4. stop

B. Help __________ on with it.

1. not 2. can 3. me 4. to

C. Where will you ___________ ?

1. cat 2. he 3. with 4. go

Twenty such sentences were written using this controlled vocabulary. Then the sentences were programmed for the Pet with Robby Robot informing students whether the answer was right or wrong. If the answer was right, the next sentence was given. If the wrong answer was selected, Robby flashed the correct answer, drooping his antennae sadly, then the same sentence was repeated. At the end of each program, the computer shows how many of the child's answers were correct on the first try. All answers are required to be correct before the program goes on to the next sentence or story.

UNIQUENESS OF THE PROGRAM

The unique and fun feature of the program is Robby Robot. Robby was created to tell children whether their answers were right or wrong. Robby is the children's friend from outer space who is so thrilled when the child selects the right answer that he smiles and twinkles his antennae. When the child supplies an incorrect answer, Robby looks sad; his antennae droop and his smile disappears. However, he recovers quickly and helps the child find the correct answer. Students are acquainted with Robby as a friend before using the microcomputer through story telling, writing stories, songs, and puppetry. Robby Robot received his name after a class contest was held to select the best name for the friendly creature.

CLASSROOM SCHEDULING

Eight classes used the microcomputer, and each had access at least one day each week. A variety of techniques were introduced to start children using the microcomputer. Some teachers used a partner technique: one student was the doer, and one was the watcher. Other teachers showed a small group how to use it, and then one child practiced at a time. One teacher demonstrated to the whole class, and then supervised each child closely at first. Some teachers recorded the child's score for each program, and others provided charts for the child to record his own score. In some cases a student teacher was used to help children on their initial try. No matter the approach, children easily learned to use the microcomputer.

The teachers received approximately two hours in-service training before school started, and the teacher who had written the programs was available in the building for help if needed during the year.

EVALUATION

Written vocabulary tests were given at the completion of each of the Houghton Mifflin Readers and the scores were recorded. Tests used were Houghton Mifflin Basic Reading Tests, Decoding Skills, Subtest 1, "Word Recognition." Scores were very high, but were not compared to the previous years scores because the classes were no longer grouped according to ability, and two classrooms had different teachers than the previous year. A valid comparison was impossible. Also, since all classroom teachers elected to use the microcomputer, there was no control group to use for comparison. Students, parents, teachers, and administrators were pleased with the program; the local school board bought more microcomputers for the school system after they reviewed the project. The evaluators from the State Office of Public Instruction, who had supplied the funds for the project, were impressed with the project and had it featured in the "Montana Schools," Vol. 23, No. 9, pages 2 and 3, May 1980, published by the Office of Public Instruction, Helena, Montana. Possibly the best evaluation is that all teachers in the Mary Innes Building are using the kindergarten and first grade reading programs on the Pet during the current 1980-81 school year.

APPENDIX: Teacher's Guide
Cassette Programs Available for Student Use
PET Computer, 1979-80

Vocabulary in programs are coordinated with books as listed.

I. Programs to use with Houghton Mifflin Books

A. Getting Ready to Read

1. Alphabet 1: lowercase letter matches with capital, used d, f, g, m, o, i, b, w, s, a, t, e. Child sees small letter and then types capital letter.

2. Alphabet 2: lowercase letter matches with capital, uses rest of alphabet not used in Alphabet 1. Child sees small letter and then types capital letter.

3. Alphabet 3 (a): as above 2, using all letters.

4. Rebus Sentences 1*: 10 rebus sentences using 15 basal words children learned first.

5. Rebus Sentences 2*: 10 rebus sentences using 15 basal words children learned first.

6. Guess the Letter: Child guesses the letter that the computer has selected. Computer tells child if he guessed "too near A" or "too far" from A.

B. Tigers

1. Tigers 1*: 20 sentences using basal vocabulary only.

2. Tigers 2*: 20 sentences using basal and nonbasal vocabulary.

3. Tigers 3: 25 Word Matching using basal and nonbasal vocabulary. Child types number of matching word.

C. Lions

1. Lions 1*: 20 sentences using basal vocabulary only.

2. Lions 2*: 20 sentences using basal and nonbasal vocabulary.

D. Dinosaurs

1. Dinosaurs 1*: 20 sentences using basal vocabulary only.

2. Dinosaurs 2*: 20 sentences using basal plus nonbasal vocabulary.

E. Rainbows

1. Rainbows 1*: 20 sentences using basal and nonbasal vocabulary from first magazine, Green Rain.

2. Rainbows 2*: 20 sentences using basal and nonbasal vocabulary from second magazine, Red Gold.

3. Rainbows 3*: 20 sentences using basal and nonbasal vocabulary from third magazine, Blue Grass.

4. Rainbows 4: 2 stories with 3 comprehensive questions for each. Uses basal and nonbasal vocabulary of entire book. Child types number of correct answer.

5. Rainbows 5: same format as Rainbow 4.

F. Signposts

1. Signposts 1*: 20 sentences using basal and nonbasal vocabulary from first magazine, Round About.

2. Signposts 2*: 20 sentences using basal and nonbasal vocabulary from second magazine, Winding Roads.

3. Signposts 3*: 20 sentences using basal and nonbasal vocabulary from third magazine, Golden Highways.

4. Signposts 4: Hangman game, words from all books. Child guesses a letter of a word picked by computer.

II. Programs to use with other reading materials in first grade.

A. Primer Seat Work

1. Primer Seat Work 1*: 20 sentences using vocabulary of entire book.

B. First Reader Seatwork

1. First Reader Seatwork 1: 12 paragraphs for vocabulary review and comprehension. Words from entire book. Child types number of correct answer.

C. Puzzle Pages

1. Puzzle Pages 1*: 20 sentences using vocabulary of entire book.

2. Puzzle Pages 2: 12 paragraphs for vocabulary review and comprehension. Words from Puzzle Pages 1 and 2. Child types number of correct answer.

D. Number Fun

1. Number words 1: 20 examples to count and read the word indicating how many. Number words include zero through ten. Child types letter of word telling how many.

2. Number Words 2: 25 examples to count and read the word indicating how many. Number words range from zero to twenty. Child types letter of word telling how many.

3. Guess the Number between 1 and 100: Child guesses the number selected by computer. Child is told if he is too high or too low.

* One word is missing in each sentence. Child types the number of the correct word. He knows immediately if he is right, or is given the correct answer to use if he misses. He must then use the correct answer before going on to the next sentence.

A STUDY OF PRESCHOOL CHILDREN'S USE OF COMPUTER PROGRAMS

Coleta Lou Lewis

Lamplighter School

Introduction

There is a computer culture in elementary classrooms. The Logo language allows the child to feed the computer, be in control, and use the computer to teach solutions to problems as the student interprets them. Cognitive knowledge is assimilated by mental representation and action. Children actively structure the world as they perceive it.

Application of the Logo language for preschool children presented an opportunity to determine how to use the language for nonreading students. Major emphasis was placed upon development of procedures that children would enjoy, the children's use of the keyboard, the relationships of the procedures to classroom activities, and use of cue cards. A study was conducted to evaluate the students' use and acceptance of four procedures written in Logo.

Procedures

Computer programs were developed in the spring of 1980 for three and four-year-old preschoolers attending the Lamplighter School Incorporated, Dallas, Texas. The private school is located in an upper-middle class, Caucasian area of north Dallas. The Texas Instruments 99/44 home computer with special extended memory and the language Tilogo, a dialect of Logo, were chosen for the project. The computer language Logo was developed by members of the Massachusetts Institute of Technology Logo Laboratory for children to learn computer programming, problem-solving, and mathematics. The computer understands Tilogo language that is typed in and follows the instructions. The built-in instructions are called primitives. The student can teach Tilogo new instructions by using these primitives.

In his article, Perlman (1976) suggested minimizing the amount of work the young or physically handicapped child must do to get an effect on the graphic screen. Special procedures were written for preschool children so they could obtain the effect of primitives by typing one key on the computer keyboard. The procedures were designed to foster the relationship between using a computer and classroom activities (drawing, blockbuilding, games with motor vehicles, relationship of numbers, speed, color, direction, space, object permanence, sequencing, classification, and visual discrimination).

Each child's intial introduction to the computer and the procedures is through the classroom teacher. The child types the procedure's name from a cue card and manipulates the variables of each procedure by typing one key. The procedures for the younger children have fewer variables; that is, they have a choice of five speeds and five colors. For the older children, ten choices of color and speed variables are possible.

The People procedure allows students to assemble on the television monitor five body parts for four people. The student manipulates color and direction by using one key for each variable. Each body part can be moved to any part of the screen and left there by typing one key. The student can continue building people until all 20 parts have been used or discontinue the procedure at any time. After the student uses all parts or types one key telling the computer they are finished, another name can be typed on the keyboard.

A computer procedure called Park allows the student to select a garage or a car shape. After the proper key is typed, a car or garage appears in the middle of the screen. Arrow keys allow the child to move each shape in four directions and change the color by typing the correct key. The child can choose any ratio of cars and garages until the total number of shapes equals 32. The shapes are designed in such a manner that both objects are visible when the car is placed inside the garage. The student can continue the Park procedure until all 32 shapes are used, in which case the computer tells the child it is out of shapes or the child can stop the procedure by pushing one key on the keyboard.

Speed, color, and direction are the input variables for a procedure called Dallas. The student can call up a truck or an airplane shape by pushing a key that corresponds to an illustration on the cue card. The speeds vary from no speed, to equal intervals of speed, to five or ten different speeds depending upon the age of the child. The procedures are written to allow the student to change speed, color, and direction variables by typing one key. The variables can be selected in any sequence for an unlimited number of times.

The fourth procedure the children can choose is called Build. The preschoolers design and create structures by manipulating squares on the graphic screen. The student controls color and direction of movement by typing one key. The procedure allows the student to build with 32 shapes.

The four procedures were used in a preschool classroom in the spring of 1980. The procedures were written in March and used with the preschoolers for 18 days in April and May of 1980. Daily records were kept of each student's experience. The data consisted of the amount of time spent on the computer and the procedures the children chose. Each child could reserve computer time by placing a name card on a chart. The student was free to use the computer for any length of time each day.

Results

The analysis of the students' time on the computer describes the total time spent on the computer by the two groups of preschool children, the amount of time spent on each program by each age level, and the amount of time spent on the computer daily by each group (see Table 1).

The three- and four-year-old children at Lamplighter School spent 1595 minutes on the computer in 18 days. The four-year-old children spent 1039 minutes on the computer during 11 days while the three-year-old children spent 556 minutes during seven days. The younger children attended school two days per week while the four-year-olds attended school three days per week.

An average of the total hours indicates four-year-olds spent an average of 94.5 minutes per day on the computer and the three-year-olds spent 79.4 minutes per day on the computer. All preschoolers attended school approximately 3.5 hours per day. One hour per day was spent on the playground.

The preschool children spent 48% of the total computer time on Dallas, 26% on Build, 15% on People, and 11% on Park (see Table 2). The four-year-old children spent 51% of the total time for their age level on the Dallas procedure, while the three-year-olds elected to use Dallas 44% of the time. The procedure Build was selected the second most often: Four-year olds used Build 26% of the time while three-year-olds used Build 25% of the total time. A larger percent of the three-year-olds elected to use People as compared to the four-year-olds. The younger children selected People 19% of the time while the older children selected People 12% of the total time. The four-year-olds and three-year-olds selected Park 11% and 12% of the time respectively.

The average number of children using the computer in a single day was seven and the lowest was two. The day two children used the computer was the first day it was in the room.

The average number of three-year-old children using the computer during the seven days was 4.28 per day. The maximum and minimum number of three-year-olds was seven and two respectively.

Conclusion

The study finds no major difference in the way the computer is used by the four-year-old and three-year-old children. Both age groups chose to spend more time on the Dallas procedure; the least amount of time was spent on the Park procedure.

The purpose of the study is to describe how four- and three-year-old children elect to spend time on four specially designed preschool computer procedures. Both age levels select one of the four procedures 48% of the total time while the three remaining procedures share 52% of the time in various proportions. Conclusions indicate that further research is needed to determine the cause-effect relationship between individual procedures and the frequency each procedure is selected. Before educators can decide what type of computer experiences to provide for children, many questions need to be answered. What is the social, emotional, and cogntive effect of various types of computer procedures? What type of computer procedures affect problem solving skills? Are there limits to the number of variables the very young child can successfully manipulate? What concepts are formed or altered as a direct result of computer experience? What effect will the child's cognitive skills have on the amount of time the child elects to spend on the computer? Will the child's cognitive skills effect the computer programs he/she chooses to do on the computer? Will the amount of time the student elects to spend on the computer be indicative of the rate of cognitive growth? Again, the question is not whether children will or will not use computers, but what will the children do with the computer? What will be the effect of using, computer on each child's development? What can we do as educators to make that effect advantageous?

Many of these questions can be investigated in a study of the comparative effectiveness of the use of specially designed preschool computer programs, their cognitive development, and the amount of time each child chooses to work on the computer.

Reference

Perlman, R. Using computer technology to provide a creative learning environment for preschool children (Logo Memo No. 24). Cambridge, Mass.: Massachusetts Institute of Technology, Artificial Intelligence Laboratory, May 1976.

Table 1
Minutes Spent on Computer Procedures
by Preschool Children

	Procedures				
	People	Park	Dallas	Build	All Programs
Four year olds	120	115	528	276	1039
Three year olds	110	68	238	140	556
Both age levels	230	183	766	416	1595

Table 2
Percent of Total Time on Each Computer Procedure
by Preschool Children

	Procedures			
	People	Park	Dallas	Build
Four year olds	12	11	51	26
Three year olds	19	12	44	25
Both age levels	15	11	48	26

PROJECTS SUPPORTED THROUGH NATIONAL NETWORKING

Sponsored by EDUNET/EDUCOM

Chaired by Paul S. Heller

Carolyn Autrey-Hunley
Elizabeth R. Little

ABSTRACT: Electronic Mail and Microcomputers

Paul S. Heller, EDUNET-EDUCOM, Box 364,
Princeton, NJ 08540

Computer-based mail and conferencing systems are beginning to be widely used for both intra- and inter-campus communications Experiences with several different mail and conferencing systems will be described and contrasted. A microcomputer, when equipped with file transfer capability such as the Apple/EASy system, can simultaneously enhance functionality and substantially reduce the cost of a mail or conferencing system.

ABSTRACT: Text Processing Network at Stanford

Carolyn Autrey-Hunley, Stanford University,
Stanford, CA 94305

Recently Stanford has undertaken a thorough study of the future of computing at the institution. A clear recommendation of the study is the development of an integrated campus-wide computer information systems network. High-speed digital communications will be provided using coaxial cable technology. A primary application of the network will be text processing, electronic mail and messaging, computer conferencing, computer-based typesetting, and electronic publishing.

ABSTRACT: Using EDUNET to Promote Literacy Among Faculty

Elizabeth R. Little, Swarthmore College,
Swarthmore, PA 19081

Swarthmore College has just substantially increased the computing capability available to its faculty, allowing many departments to introduce or expand the use of computing in the curriculum. EDUNET has been and will continue to be used by faculty to sample high-quality instructional and research programs. This experience makes it possible to decide rationally between 1) local development of software, 2) acquisition and installation of software from outside sources, and 3) continued use through EDUNET.

GUIDELINES FOR THE SUCCESSFUL SELECTION AND OPERATION OF MINICOMPUTERS AND MICROCOMPUTERS

Douglas S. Gale
Decentralized Computing Services
Cornell University
Ithaca, NY 14853

ABSTRACT:

This tutorial is designed to assist users select, acquire, and operate minicomputers and microcomputers in a laboratory or classroom. The tutorial is designed for the relatively unsophisticated computer user and covers the following general areas:

History of small computers

Relative advantages and disadvantages of micro/mini/midi and maxicomputers

How to determine what kind of computer is best for you

A brief survey of available hardware and software

How to select a small computer system

Warranty, maintenance, and operating costs

COMPUTER-BASED EDUCATION: STRATEGIES FOR SUCCESS

Franz E. Fauley
Computer Based Education Systems
13 Acorn Drive
Hawthorn Woods, IL 60047
(312) 438-8271

ABSTRACT:

Professional educators and trainers concerned with the development of others must be ever alert to new technologies that make the mission of training and education more efficient. Although CBE promises to do exactly that, many educators find it difficult to justify the use of computer-based training in business and industry. This session gives a rationale for defending the use of CBE in the corporate environment and outlines the requirements for successfully establishing CBE. It will review five major criteria for selecting a subject that will be well received by trainees and management alike.

Since one of the first major decisions that must be made about CBE concerns the issue of courseware delivery, Mr. Fauley will conclude his presentation by reviewing the functional and economic differences that exist between stand-alone and centralized CBE systems.

Session Outline:

I. Encouraging the Use of CBE
   A. A review of the need to establish stronger ties with modern technology.
   B. The impact of exponential growth on corporate training.
   C. Employee growth rates in business and industry.

II. Establishing a Successful Corporate CBE Effort
   A. Gaining top management commitment.
   B. Integrating your CBE system with other computer-dependent systems.
   C. Building team effectiveness.

III. Determining the Criteria for Subject Selection
   A. Stability.
   B. Use by a wide audience.
   C. Logistics and geography.
   D. Trainee/instructor interaction.
   E. Demonstrable cost savings.

IV. Selecting the Right Delivery Mechanism
   A. Stand-alone CBE systems.
      1. Major characteristics.
      2. Advantages
      3. Disadvantages
   B. Centralized CBE systems.
      1. Major characteristics
      2. Advantages
      3. Disadvantages

PRECOLLEGE SOFTWARE EVALUATION AND ADMINISTRATION

Karen Billings
Daniel H. Watt
Ted Mims
Charles G. Boody

ABSTRACT: A Joint Project

Karen Billings, Microcomputer Resource
Center, Teachers College, New York,
NY 10027

The EPIE Institute and the Microcomputer Resource Center at Teachers College, Columbia University, have jointly undertaken a project to analyze a selected group of microcomputer curriculum products. The purpose of the analyses is to provide instructional leaders in participating schools with consumer product information to help them select and purchase microcomputer materials that fit the needs of teachers and learners.

The project will first produce updatable files that contain comparative analyses of the microcomputer courseware produced by major publishers. The first stage looks at only the microcomputer software that covers a wide range in the school curriculum.

An analysis instrument stresses the aspects of good instructional design specific to the new technology. It looks, through a number of different levels, at the intent and content of each program, the methodologies used, and the means of evaluation. Design congruence and use considerations help summarize each analysis.

The analyses will be most helpful to school personnel at the district and the education service agency level, such as curriculum supervisors, department coordinators, and media specialists.

ABSTRACT: A Description of an Educational Computer Resource Center

Daniel H. Watt, MIT, Brookline Public
Schools, Technical Education Research
Centers, Boston, MA 02139

During the spring of 1980, Technical Education Research Center, a non-profit educational research and development firm in Cambridge, Massachusetts, established the Computer Resource Center to provide educators throughout New England with up-to-date information about instructional uses of computers in pre-college education. The center includes a representative collection of different microcomputers, books and periodicals relating to educational computing, and a range of software for demonstration purposes. Activities at the center include a drop-in microlab for educators, workshops for teachers and school administrators, consulting with school systems, and the early stages of a computer education intern program and a cooperative software evaluation exchange.

This presentation will describe the center's activities and discuss the strategies we are using to develop a supportive community of center users and to create an ongoing base of financial support. We will offer an appraisal of successes and failures during our first year of operation, and hope to spark discussion and exchange of ideas with other groups engaging in similar ventures throughout the country.

ABSTRACT: School Scheduling System

Ted Mims, Department of Computer Science,
Louisiana State University,
102 Nicholson, Baton Rouge, LA 70803

The School Scheduling System (SSS) is a complete, integrated scheduling and record keeping system for elementary and secondary schools. SSS incorporates simple and easy-to-follow menu directives requiring minimal operator training. The system is user-oriented and contains many features to reduce operator error, along with extremely powerful error recovery procedures. School administrators and teachers were consulted and assisted in the development of the system. SSS produces several reports and maintains student and school personnel files. These files are designed with flexibility to allow for modification and expansion. The programs used by SSS are modularly designed to allow for additional expansion as the need arises.

ABSTRACT: Comprehensive Achievement Monitoring: Instructional Management on a Microcomputer

Charles G. Boody, Hopkins School District,
Administrative Offices, 1001 Highway 7,
Hopkins, MN 55343

Comprehensive Achievement Monitoring is an instructional management tool that can produce information upon which teachers can base teaching and curriculum decisions, and with which some accountability questions can be answered. Whem CAM is used, teachers are expected to:

1. Define their course with behavioral objectives.
2. Develop test items to measure student performance on each objective.
3. Build a set of randomly parallel tests, each of which evaluates a sample of the course objectives.
4. Design a regular, usually bi-weekly, program of testing the students.
5. Provide for the computer program the date on which a student is taught each objective.

The Hopkins, Minnesota, School District has been involved with CAM developments since 1969. During the past ten years, they have received two federal grants to aid development and dissemination of their work. In 1978, they contracted to develop CAM on the Apple II microcomputer, a project that has been successfully completed and widely used despite the assertions of some "experts" who claimed it could not be done. Presently, the system is in use throughout the continental United States and in Alaska. The report to NECC will include detailed information about the system's limits, the reports presently available, and the extent to which the system can be expanded.

SOME USES OF COMPUTER GRAPHICS IN THE CALCULUS CLASSROOM

Robert F. Maurer
Timothy P. Donovan
The Penna. State U.
Mont Alto Campus

INTRODUCTION

Human limitations are a constant hindrance to effective teaching in calculus classes. Lack of artistic ability and the slowness of hand computations, even with a hand-held calculator, are two of the most frustrating. Most courses require that so much material be presented that little class time is available for carefully sketching graphs or including laborious calculations to illustrate such concepts as the δ-ε definition of limits or the upper and lower sums converging to the definite integral. Too often the graphs are sketched hastily and poorly. Important concepts are glossed over or ignored altogether.

At several campuses of The Pennsylvania State University, the unique graphics and computing capabilities of the Apple II Plus microcomputer are being used to overcome these difficulties. A library of classroom demonstration programs is evolving through the cooperative efforts of several faculty.

This paper discusses some of these pedagogical situations and the programs that have been written to deal with them. All the programs discussed were written in Applesoft Basic, a floating-point Basic for the Apple II Plus. Basic is quite easy to learn for a computer neophyte. Naturally, it is even easier for someone who knows Fortran. In fact, with any computer background at all, one can begin programming immediately, referring to the excellent manuals provided by Apple Computer, Inc. only for trouble spots.

SKETCHING GRAPHS

Every mathematics instructor, at one time or another, has longed to be able to sketch graphs of functions quickly and accurately, expecially when discussing how similar functions are related. Several examples come quickly to mind: $y = x^2$ vs. $y = x^4$, $y = e^x$ vs. $y = 2^x$, and $y = x^2$ vs. $y = x^2 + 2$ vs. $y = (x + 2)^2$.

Applesoft Basic contains an HPLOT command which makes it very easy to write a graph-sketching routine in the high-resolution graphics mode. The particular program being used also contians a function input routine which allows the user to sketch the graphs of several functions on the same set of axes in different colors. The scales on the x- and y-axes can be adjusted by the user.

An error handling routine draws a vertical line in white at each x-value for which the function is undefined. Vertical asymptotes are thus drawn automatically, and white bands are sketched for x-intervals yielding negative arguments for the square root function. However, because Basic does not understand complex numbers, the isolated point at the origin in the graph of $y = x/\text{sqr}(x^2 - 1)$ is treated as undefined.

Improvements to the program will allow specifying the location of the axes on the screen and a particular x-interval for the graph.

DERIVATIVE DEMONSTRATION

Students understand derivatives much better when they grasp the relationship between the graph of a function and the graph of its derivative. This relationship is difficult to show without preparing transparencies or laboriously sketching graphs at the board.

Two different programs are currently being used. These will eventually be revised into a single program with several options.

The first program plots the graph of the function in the top half of the screen, and then plots the derivative of the function in the bottom half of the screen. The scales on the axes are calculated by the program and are the same for the function and the derivative. A short tangent line follows along the graph of the function as the graph of the derivative is being plotted. To illustrate critical points, vertical lines are drawn at the zeros of the derivative from the derivative x-axis to the graph of the function. The derivative is graphed using approximation to the slope of a secant line with a very small value of Δx, e.g., 0.001.

The second program plots the function on the full screen and then plots the derivative on the same axes in a different color. Higher order derivatives can also be plotted on the same axes. This program can optionally demonstrate the definition of the derivative as the limit of secant lines. The user moves a point along the curve using the game paddle. Pushing the button on the paddle fixes the point. After the initial point is fixed, pushing the paddle button causes a line to be drawn through the current point and the fixed point. This process can be repeated, each time moving the second point closer to the fixed point.

POLAR GRAPHS

When teaching functions in polar coordinates, the instructor encounters two basic difficulties: the graphing is time consuming, and the traditional approach is inadequate for anything more complicated than a simple rose or cardioid. To compound these problems, the student feels uneasy with polar coordinates of any sort, making it an imperative to provide many detailed examples.

Using a polar coordinate sketching program, the instructor can display many graphs in the time it takes to do only one by hand. The student can see what difference a change in the function can make in the graph. Of course, it is still a good idea to do several graphs by hand before the computer demonstration so that the students can see what the program is doing and gain a real appreciation for the speed and clarity of the Apple II's graphics.

SURFACE SKETCHING

Anyone who has taught three-dimensional analytic geometry knows the difficulty in producing a graph of any sort for these surfaces. The program sketches surfaces of the form $z = F(x,y)$ by sketching level curves for constant x. Surfaces of the form $z^2 = F(x,y)$ can also be done by first sketching the positive square root and then the negative. The "bottom" of the surface as seen by the viewer can be sketched in a different color.

The sketching process is somewhat slow. However, using a floppy disk allows the graphs to be plotted in advance and rapidly recalled in class. Using the disk, many graphs can be shown and discussed in the time that it takes to generate one. It is probably best to show the students one graph from start to finish so that they can understand the process of plotting level curves.

DIFFERENTIAL EQUATIONS

Most ordinary differential equations courses today include a chapter on numerical solutions. The usual approach is to consider differential equations of the form $y' = F(x,y)$ with an initial condition of the form $y(a) = b$. To demonstrate the types of error involved and some of the pitfalls, some problems have to be carried out for many iterations. Such a calculation is practically impossible without at least a programmable calculator. But the Apple II displays the results as they are generated, which is faster than passing a computer printout around the room.

The program allows the user to specify either the Euler or Runge-Kutta methods of solution. It offers the latitude to select initial conditions, increment, and number of iterations printed. After displaying a table of solutions, the program will graph the approximate solution, if desired. This feature allows a visual demonstration of some anomalies as well as the role of initial conditions in the solution.

CONCLUSION

The above are only some of the obvious uses for a microcomputer in the classroom. Additional topics for which programs are being written or are contemplated include limits, the definite integral, and areas in polar coordinates. The Apple II Plus handles all of these tasks well, and in most instances has finished its task before the instructor can explain what it is doing.

While no external demonstration substitutes for effective teaching, the Apple does complement instruction well. The students relish the change of pace that it offers. The advantage of providing the borderline student with one more explanation for a difficult subject should not be overlooked. The only problem with using the Apple is the amount of time the demonstrations use. But with careful planning, total class time can actually be reduced. The students are comfortable with the television presentation of the Apple. During the demonstrations they usually express an interest in the program itself and often suggest problems that they would like to see. Several students have indicated that they understood the material much better after seeing the computer graphics.

In conclusion, while the idea is not perfect, the authors believe that the experience for both students and faculty has been a highly positive one.

THE HUMAN FACTORS OF COLOR DISPLAY-BASED INSTRUCTION

John Durrett, Catherine Zwiener and Robert Freund*

Center for Automated Systems in Education

Southwest Texas State University

INTRODUCTION

The Center for Automated Systems in Education is located at Southwest Texas State University. The mission of the Center is to explore the human factors of automated systems in instruction. The Center conducts validation studies of computer-based instructional systems and human factors studies associated with automated systems. We are currently conducting research on the use of color graphic displays for computer-based instruction under the sponsorship of the Control Data Corporation. Color displays are becoming more common and are more complex from a human factors perspective than black-and-white displays. A good working knowledge of the human factors of such displays is essential for their effective use.

Many individuals who use color display systems want to know what colors should be used. Many combinations should not be used. The characteristics, capabilities, and limitations of the human information processor that will be viewing the display must be considered. Interaction with a color display is dependent on perception. It is through an understanding of the factors that affect our perception that we can obtain guidelines for the use of color displays.

PERCEPTION AND THE VISUAL SYSTEM

Obviously, the nature of the stimulus and the structure of the visual system affects our perceptions. This fact provides the empirical foundation for statements and recommendations about using color displays.

Structure of the Eye

The eye focuses light from external objects on a light-sensitive area known as the retina. The retina, the most complex component of the eye and an extension of the brain, is composed of many types of receptors two of which are rod- and cone-shaped cells. Rods respond to low levels of illumination and produce visual sensations of shades of gray, but no color. Cones respond to high illumination and produce visual experiences of color and detail. Rods produce their maximum response to yellow light. So, the relative brightness of matched blues and reds and greens and reds varies as a function of ambient illumination. In fact, if green is to be as vivid as red under high illumination it must be three times as bright. Changes in illumination cause cones to adapt after about 7 minutes. Rods require about 30 minutes to totally adapt.

One of three different color pigments present in a cone makes it sensitive to red, green, or blue light. Cones also differ in the level of sensitivity to a particular color of light. The cones are completely active at 435nm for blue, 535nm for green, and 565nm for red. Blue receptors are significantly less sensitive than the green or red receptors.

The theory that explains how we perceive color from these receptors postulates an opponent process mechanism. Three opponent receptors--blue-yellow, green-red, and white-black--produce our color sensation by various increases and decreases in neural firing rates. Current theory emphasizes adaptation, contrast, color appearances, and afterimages to explain color vision. For example, since it is impossible to see a mixture of red and green in the same patch of light, these sensations are explained as results of opposite and incompatible activity in the same system.

Near the center of the retina is a slight depression about 1 sq mm known as the fovea. The fovea is composed entirely of cones. These cones are responsible for detailed vision processes. The concentration of cones decreases moving away from the fovea to the periphery of the retina while the concentration of rods increases to a maximum in the periphery. Since high concentrations of cones produce detailed visual experiences, visual acuity decreases as the distance from the fovea increases. There are zones on the retinal surface that are differentially sensitive to different colors. The eye is sensitive to all colors at the fovea; reds and greens become harder to perceive toward the periphery, then yellows and blues become more difficult to perceive. There are no cones in the periphery and thus, no colors are perceived in that area.

---

* This material is based upon work supported by the Control Data Corporation under grant # 80609.

These limitations of the visual system lead to the following recommendation about color display organization. Since red and green areas of the spectrum are reduced at the periphery of the visual field, do not use red and green outside of normal line of sight or place codes in these colors where they are likely to be overlooked. If they must be used at the periphery of the visual field, make them blink first before continuous display to get the user's attention.

Color Perception

Color vision is a complex process of three interacting psychological parameters: hue, brightness, and saturation. Hue is what we normally think of as color. Brightness is related to the intensity of light reaching the retina. Generally, higher intensity light sources appear brightly colored while lower intensity light sources appear more dull. However, the retina is also differentially sensitive to various wave lengths in the spectrum. Yellow is perceived as the brightest spectral color while red and blue are perceived as the least bright. Saturation is produced by the interaction of hue and brightness and is that aspect of color most strongly influenced by the addition of white light. For example, a 100% saturated spectral red becomes more pink with the addition of white light. However, in terms of hue it is still red, only a red of decreased saturation.

Contrast

In addition to these psychological components, there is the human dependent parameter of contrast. While brightness is essentially a measure of light intensity of a signal, contrast is the relative brightness of signal over background. The greater the difference in wave lengths between two colors, the greater the contrast and readability of a display. Darker colors such as red and blue are not as visible as lighter colors such as white or yellow when viewed on a dark background. Using higher contrast and colors that have large differences in wave lengths produce a more readable graphic. This is not a subjective phenomenon but arises from the characteristics of the visual system.

Reductions in contrast deminish our ability to determine details. Research has indicated that visual acuity depends on the colorof the symbol and the type of background. In fact, symbol size must be increased from 15 to 45 minutes of arc as the number of colors increase from 1 to 7 for adequate color perception.

Task Environment

The visibility of a display is also affected by the task environment. High ambient illumination reduces symbol to background contrast and lowers visibility. Further, sensitivity to color increases as the visual system adapts to darkness. Improper illumination of the color display environment can result in reduced performance, discomfort, and subjective fatigue and limits the effectiveness of color changes. Ambient illumination can result in reaction time changes for various colors. Further, reaction times are fastest for red and blue while they are slower for yellow and yellow-orange. Depending upon the application, these may be important variables to consider when using color in a display.

A general conclusion is that if high illumination exists, red should be used as the key coding color. Further, use of blues and greens should be limited to large zones or lines and not used for alphanumerics. Also, blues should not be used for coding information presented in the fovial region. Based upon the properties of the visual system, suggested colors for coding are with yellow, red, and white.

Color Deficiencies

Finally, we must recognize that not all individuals have a perfect visual system. Between 6 and 10% of the male population has defective color receptors and are unable to see certain colors. Less that .05% of the female population is defective in color vision. Color-based coding should not be used when individuals with color defects such as color blindness will operate a display.

PERCEPTION AND LEARNED EXPERIENCES

A second factor that affects our perception is our learned experiences. This factor also has implications for the use of color displays and is a complex area in which behavior is determined by past learning and the inherent human processing capabilities.

Limits of Processing

Research has generally indicated that a limited number of color codes should be employed in most contexts. However, with training, as many as 50 colors have been employed in the laboratory. In general 7 +/- 2 colors can be employed with experienced, long-term users.

Attention

Color influences attention. By carefully utilizing color to manipulate attention, the user can partition material at key points, organize it, and code it. Techniques that direct attention increase the likelihood that the attended to information will be processed.

Learning and Comprehension

Color can be a significant influence on learning and comprehension of material presented on color displays. Color is superior to black-and-white presentations because it speeds processing time and organizes and motivates the learner. But no differences in the interpretation of information are observed if adequate study time is allowed. Information presented in color is recalled better than material presented in black-and-white for some individuals. Further, color can assist learning if used as a redundant cue or to highlight key concepts. However, the color of key concepts and responses must be matched for maximum performance.

Population Stereotypes

Consider carefully population stereotypes when utilizing colors to present information. For example, among colors available for color graphic displays, the colors red, green, and yellow have been stereotyped in the population to denote "down", "stop" or "danger" for red, "up", "go" or "ok" for green and "wait", "caution" or "slow" for yellow. The application of color to a specific task should employ these color stereotypes to achieve maximum performance. Graphics that use red and green in ways contrary to the stereotype can interfere with information processing and result in incorrect conclusions. While graphics that use red and green in ways that agree with the stereotypes can assist information processing. Colors should be used in ways that agree with the text.

PERCEPTION AND INDIVIDUAL ATTITUDES AND VALUES

The final factor affecting our perceptions are our individual attitudes and values. This factor also influences our use of color displays. Quite honestly, people prefer color displays to monochromatic displays for several reasons. First, there is less subjective fatigue. This is certainly one reason green phosphor displays are less subjectively fatiguing than black-and-white displays. Color also motivates. As we have noted earlier, color is a powerful attention manipulator that can be entertaining and aesthetically pleasing.

In summary then, any use of color to portray and convey information should employ maximum wave length separation to produce maximum color contrast. Careful attention must be devoted to contrast variables to assure legibility and reading ease. Always use highly saturated colors. Limit the number of colors used to four. Remember, the limitations of the human information processor are more severe than the limitations of the hardware. Always code alphanumerics in red, white, or yellow while limiting use of blues to large non-fovial areas. Code peripheral signals in white since red and green are not easily visible at the periphery of the visual field. And finally, follow conventional color uses. Knowing and using these findings will result in improved performance of individuals using color displays.

BIBLIOGRAPHY

1. Durrett, H. J. Color display systems: State of the art. Behavior Research Methods and Instrumentation. April, 1979, 11.

2. Durrett, H. J. and Zwiener, Catherine. Human information processing and color displays. Proceedings of Southcon/81. January, 1981.

3. Neil D. E. Design parameters and color CRT display design. Naval Post Graduate School, Monterey, California. February, 1979.

GRAPHICS
IN
COMPUTER SCIENCE

Freeman L. Moore
Computer Science Department
Central Michigan University

ABSTRACT

Many course offerings have been influenced by Curriculum '68 and its updated version of Curriculum '78. Curriculum '78 proposed a special topics course to include new subjects. One such topic is computer graphics. With the strong interest in graphics, non-engineering computer science departments should consider such a course. Some of the problems and limitations of a course are described. Exposing the student to several software packages and hardware items is presented and discussed. The course is limited by hardware, although a great deal of information can be presented with minimal hardware support.

Introduction

Curriculum '68 [3] and Curriculum '78 [4] have been beneficial to computer science departments for the formation of their current course offerings. Central Michigan University has been no exception; our course selection has been extended in several areas since the early 70s [5]. As a result, we have chosen to offer a semester course in computer graphics on a regular basis, rather than as a special topics course as Curriculum '78 suggests. While this may not be considered suprising, you must note that this department does not emphasize engineering. Our students, undergraduates and graduates, are more interested in software. Students receive exposure to hardware via logic circuit design and microcomputer courses. But they are exposed to several languages and concept courses. Our students have been highly esteemed by numerous employers for their thorough background in various software components. Computer graphics represents one aspect. The topic of this paper is the problems incurred when teaching a graphics course with a non-engineering emphasis, and limited hardware.

Need

All of us recognize computer science as a multidisciplinary field, and that certain key components can be stressed. One expanding area is computer graphics, as evidenced by the growth in the special interest group on computer graphics (SIGGRAPH). The attendance at SIGGRAPH 80 notably exceeded the attendance at ACM 79 [8]! With such a massive interest group, computer science departments need to devote attention to this growing area. Graphics has been historically linked to engineering departments and CAD/CAM work. Areas such as CAI, animation, visual display of data base links, etc. are just as important and deserving of investigation. But the study of computer graphics has certain appeal since the results are visible in a different manner than the normal computer printout. Students are notorious for using computers to play games and draw pictures, producing results which can be enhanced with the aid of graphic techniques. Even the current microcomputers have greater graphics capabilities, making possible simple animation and graphics.

Approach to Teaching Graphics

Any course in computer graphics is a multi-faceted course. Topics from mathematics, operating systems, algorithm design, logic circuit design, and others are introduced. In our course, we consider ourselves as end-users of software packages. To understand the operations of the final product, some time must be spent looking at the lower-level of the software. So, introductory time is used to discuss the special control codes for various graphics devices, such as a plotter and a storage tube display, e.g., CalComp 565 plotter, Tektronix 4013 graphics terminal. It is doubtful that the bulk of the students will design graphic systems, but they may have the opportunity to incorporate the ideas in software systems. It is essential for them to understand that they are only operating at the tip of the iceburg with the software. An outline of the course as taught at Central Michigan University is given in Appendix A, which illustrates the mixture of topics presented in the discussion of graphics.

Software

To teach graphics, the necessary software must be made available to students. In our situation, the department owns the graphics equipment and has the responsibility of maintaining the software. This differs from the other situations where the computer center is in charge. We make available the Basic CalComp [1] and Tektronix Plot-10 [6] packages for public use. Since these packages are the most widely accepted, we hope students understand what to expect in other commercially available packages. One area which both software libraries lack is operations on three-dimensional information. This must be compensated for in the classroom, but it provides the opportunity to develop these and other algorithms for data manipulation, including perspective drawings of stick-figures.

While students are expected to become proficient with the equipment available on campus, classroom discussion includes hardware features not available as well as other software packages. The special issue of SIGGRAPH [2] has proven to be an excellent reference, and is assigned as a text, along with the text by Newman and Sproull [7].

One advantage of the SIGGRAPH publication is that it provides a description of the core-system, proposed as the standard for computer graphics, This introduction gives students the needed background information should they come into contact with the core-system at a later point. The concept of standardization of software packages is a novel idea for students, although they thoroughly appreciate the standards for programming languages.

Interactive Computer Graphics

Any course in computer graphics should include time working with a plotter, be it a drum or flatbed. Installations will often have some form of a plotter (or even a line-printer) before they will have a graphics terminal. Plotters are ideal for the types of drawing they produce, but the construction of interactive plotter programs is often difficult. Most operating systems consider plotters as off-line equipment, or more likely, the plotter is not next to the terminal where the person is interactively executing a program. For these reasons, plotter programs are often batch-oriented. An alternative to this is to develop alternate run-time libraries that can be loaded at run-time without changing the source program. The goal is to allow the same program to display its results on a different device with only a change in the job control language. One such package available is PPF, which allows a choice from several libraries [9]. For our use however, we developed a utility for our CalComp software to allow a user to preview their plot file. This is not completely interactive because our version is not included at run-time, but is merely a separate program which can read a created plotfile and display it on our Tektronix terminal. This is a relief for our computer operations staff, mainly because of the reduction in the number of plots sent to the plotter. Students enjoy it as well because of the increased availability of the Tektronix terminal as opposed to the limited times the plotter was operated.

By the time students register for CPS 575 - Computer Graphics, they will have completed Fortran programming and PL/I programming, in addition to a mathematics course on matrix operations. The interesting aspect of their background is that they have not learned the art or science of developing interactive programs. The above mentioned programming courses stress the language. The students may have executed their programs at a terminal, but the approach was to provide a batch-style output display and input format. In the area of graphics, the word "interactive" plays a key role when dealing with devices other than the plotter.

Certainly programs are easily identified that are not interactive with the user, but defining requirements for interactive programs is much harder than for batch. For our purposes, we assume the user is not reading a user's guide while sitting at a terminal. The program executed at the terminal must lead the user, prompting for needed information in a clear and self-explanatory fashion. A common mistake is to have the user enter 0 or 1 when the intended response is YES or NO. Students are limited to one side of one sheet of paper that must include a program abstract and the operating system commands needed to execute the program. Once the program is started, we assume that the paper is no longer used. This approach has been awkward for some students. While understanding their own program, they may have difficulty conveying its operations to others. Students are encouraged to solicit volunteers from another class to test the written directions for clarity, simplicity, and correctness.

Concepts versus Algorithms

After stressing for interactive programs, concentration is placed on the role of computer graphics and what it can do. In some areas algorithms are presented, for example, for drawing straight lines between two points on a plotter or on a microcomputer display. But the most part, students need to be aware of what the software can do, not necessarily how it is done. With emphasis on concepts rather than algorithms, more useful information can be covered, including a survey of other graphic packages. For the student who expects to be a graphics software designer, references to various algorithms are included, with attention also drawn to the text for details and theory.

Low Cost Equipment

To consider teaching a course on computer graphics, a minimal amount of equipment is needed. Expensive graphic terminals like an IMLAC can be substituted by a lower cost storage-tube display terminal such as a ektronix, or even an inexpensive microcomputer. Microcomputers such as Apple are attractive for their color displays. Large plotters can be replaced by older and smaller models, such as the model 565 we use. A plotter is now available for use with microcomputers, costing less than $1,000. One could also consider a dot-addressable printer such as Printronix. Alternatives to expensive main-frame equipment exist and should be considered.

Conclusion

The students have received practical experience working with a plotter, employing a Tektronix graphics terminal, and using cross-hair and graphics tablet for input. The TRS-80 has been used to illustrate the concepts of raster graphics, as well as light-pen input. Better examples of raster graphics need to be done, but the TRS-80 does provide some insight. While we don't have the best (or much) of equipment, we have provided students with a very rich and rewarding learning experience in computer graphics.

Computer graphics is not a discipline solely restricted to computer science departments. Mathematics and the other sciences (chemistry, physics, geography, etc.) use our equipment and students for their needs. This kind of acceptance is rewarding and demonstrates that graphics can be a valuable tool.

BIBLIOGRAPHY

[1] CalComp Software Reference Manual, California Computer Products, Inc. Anaheim, CA., 1976.

[2] Computer Graphics, ACM, June 1978.

[3] "Curriculum '68", Communications of the ACM, March 19, '68.

[4] "Curriculum '78", Communications of the ACM, March 1979.

[5] "A First Course on Files", F.L. Moore, SIGSCE Proceedings, February 1979.

[6] PLOT-10 Terminal Control System User's Manual, Tektronix, Beaverton, OR., 1976.

[7] Principles of Interactive Computer Graphics, Newman & Sproull, McGraw-Hill, 1979.

[8] "President's Letter", Communications of the ACM, August, 1980.

[9] Pascal Plotting Facility, University of Illinois, 1979.

Appendix A

CPS 575 Outline

Introduction: 1 week

Explain the need for graphic devices and output. Include examples from current video games to more complex examples of radar simulation and flight simulators.

Plotter Technology and Software: 3 weeks

Begin by introducing the physical capabilities of a drum and pen plotter, using various hardware codes. On top of this, include the software of CalComp, with the graphic primitives, as well as advanced routines for scaling, drawing symbols, displacement, and rotation of objects. Scaling of data to fit an axis is also introduced.

Tektronix Technology and Software: 4 weeks

Start by comparing the software of the CalComp with the Tektronix, identifying operations not available or not equivalent. Tektronix has input capability via the cross-hairs, thus introducing windows, and the concept of clipping. Algorithms for clipping are presented, with the PLOT-10 software performing the operations on user defined windows. Discuss the role of alphanumeric output as well as graphic output to a single device. Menu selection for game playing is also introduced.

Feature Comparisons: 1 week

After having spent time learning the CalComp and PLOT-10 software, some time is devoted to discussing other systems, including DISSPLA, GINO-F, and the core-system. Techniques and guidelines for evaluating software packages as well as hardware are presented.

Transformations: 3 weeks

To complement existing software packages, 3D operations are introduced and 2D operations reviewed, both in the algebraic fashion and matrix form. Perspective is included as a mapping from 3D onto 2D. More operations on menus and window selection are included here.

Graphic Devices: 2 weeks

Since graphic devices generally input as well as output, discussion of other hardware is included, such as tablet input for 2D and 3D, raster operations, light-pen input. Discussion of hardware and software requirements is included. Introduction to the TRS-80 and its mode of graphics.

Surfaces and Lines: 2 weeks

The problem of removing hidden lines and performing surface rendition is introduced at this time, mainly due to lack of proper equipment. Shading and color concepts are also mentioned.

Appendix B

Programming Assignments

1. Plotter

Given as input the height and width, use this information to draw a goblet, with the cup of the goblet being an ellipse. An introductory program involving the plotting of a function.

2. Plotter

   Given a set of data containing enrollment in CPS courses for the past several semesters, provide a graphical display of the information. The objective being to use the various routines for scaling, axis, and symbols.

3. Tektronix

   Provide a menu of 4 items to choose, and then draw the selected items elsewhere on the screen. The objective is menu operations, and crosshair input in addition to the basics for CAI.

4. Tektronix

   Given a set of data representing a three-dimensional stick house, provide various views as requested by the user. The program will stress interaction with the user, and 3D algorithms, as well as an introduction to animation.

5. Tektronix

   Develop a simple calculator program by using the graphics tablet for the Tektronix as the "Keyboard". The objective is to understand tablet input, as well as to do more work with developing interactive CAI programs.

6. Open

   The students are requested to decide a topic as their last project, possibly extending one of their earlier works, or to consider using the TRS-80 microcomputer for some type of raster graphics.

MICROCOMPUTERS IN EDUCATION: A COURSE IN COMPUTER LITERACY FOR EDUCATORS

Dr. Cheryl A. Anderson
University of Texas
at Austin

INTRODUCTION

Due to the advent of the microcomputer, the computer revolution has now reached the American classroom. For the first time, the computer is cost effective, and as a result, more than 50,000 microcomputers will be in use in the classroom this year.(Manji, 1980) Parents seem very supportive of the microcomputer's use because it introduces their children to a technology upon which our society is becoming increasingly dependent. We are just becoming conscious of the fact that out children need to be computer-literate just as much as they need to learn to read and write. The responsibility for teaching children to understand and use computers will rest upon the teachers in our school systems. Unfortunately, most teachers are woefully unprepared to even begin planning a curriculum using computers. Many have never touched a computer much less programmed one. Some are even fearful. As David Moursund reported at the 1980 NECC convention, "We are asking computer-illiterate teachers to help students become computer-literate at a funtional level (Moursund, 1980, p. 128).

Many have called for the need to train teachers about the computer and its applications in education (Milner, 1979; Dennis, 1979; Taylor et. al., 1979). The Elementary and Secondary Schools Subcommittee of the ACM Curriculum Committee has published a set of competencies that represent in their words "...the scope and substance of teacher training needed to integrate computing into the schools (Taylor, Poirot, & Powell, 1980). This committee has defined competencies which are universally needed by teachers, as well as competencies needed by teachers of computer science and other specific subjects. This committee has recommended that students in teacher training be required to obtain the universal competencies and either the set of computer science or those of specific subjects. As yet there are few courses offered to future or current teachers that address the issue of computing competencies for teachers (Milner, 1979). This paper describes a course taught at the University of Texas at Austin in the College of Education which was developed to help teachers learn about computers, and specifically about microcomputers.

COURSE ORIGINS

Because of the increased interest in the use of microcomputers by educators, the College of Education at the University of Texas at Austin purchased several microcomputers which were placed in the Learning Resources Center at the College. These microcomputers are available for faculty and student use. Purchasing six TRS-80 microcomputers and one Apple II Plus microcomputer enabled the Media Education department to offer a course entitled "Microcomputers in Education." The course was offered for the first time during the summer 1980 session and again during the spring 1981 session.

The target audience for the course includes elementary and secondary school personnel, librarians, University faculty, and graduate students in Curriculum and Instruction, Library Science, and Computer Science. One reason the course is directed to this audience is that these people are often responsible for making decisions about the purchase and use of equipment. Because using microcomputers in education is so recent, they do not have the kind of experience or knowledge required to make such a decision about microcomputers. In addition, many of these people are responsible for training others, therefore, knowing hardware and software resources, as well as resources for computer-literacy materials and funding is important. Thus, the course was designed to help prepare the participants for their roles as decision-makers and trainers.

COURSE OBJECTIVES

Students are expected to set up and operate a microcomputer; develop a simple CAI program; propose and justify the purchase of a microcomputer system; and identify and select the appropriate hardware and software in accordance with their instructional needs. In addition, the course seeks to make the student more aware of the growing need to become computer-literate. More specifically, the student should be able to:

1). discuss the history of computing from the abacus to the microchip;
2). label and discuss the functions of the major components of a microcomputer system;
3). identify and discuss the uses of various microcomputer hardware;
4). compare the advantages and disadvantages of a microcomputer system versus a time-sharing system;
5). develop and discuss criteria for the selection of hardware and software;
6). compare and contrast various hardware and software, and based upon a set of criteria, make a selection;
7). identify a list of resources for hardware and software;
8). identify a list of resource materials for teaching computer-literacy, including print and non-print materials;
9). identify a list of funding resources that can be tapped for the development of educational programs that use the microcomputer;
10). identify and discuss the major educational uses of the computer, i.e. drill and practice, problem-solving, tutorial, text editing, record keeping, simulations, etc;
11). write a proposal justifying the use of the microcomputer in a particular setting, including review of literature, a budget, and a list of resources; or write a curriculum for a computer literacy course;
12). do simple Basic programming;
13). recall and describe the steps involved in the systematic planning of instructional computing programs; and
14). define common terms used in the computer sciences.

COURSE SCHEDULE

The course meets once a week for two and a half hours during the regular sixteen-week session and for one and a half hours each day during the summer session. When taught during the summer, the course content and objectives must be modified due to the limited time. It is impossible to adequately cover all the materials in just six weeks. The text books used in the course are the AEDS Journal Fall 1979 issue entitled "Microcomputers: Their Selection and Application in Education", Computers in Education by Jim Poirot, and the Microcomputer Workbook by Jim Poirot and Don Retzlaff.

The topics covered in the course are as follows:

1. An introduction to course and hands-on experience with the microcomputer: In addition to a typical course introduction, the media education department has found that the video tape "Don't Bother Me I'm Learning" (available from your Bell & Howell dealer) is a helpful introduction to the use of the microcomputers in teaching. The tape provides testimonials from teachers, parents and, best of all, students speaking of their positive feelings about computers in the classroom. Competencies are then discussed, followed by a hands-on experience with the microcomputer. This first day experience is necessary to relieve anxieties about the computer. Students learn how simple it is to load and run prerecorded programs. In addition, this experience serves as a basis for the discussion of computer components.
2. Basic programming: Two weeks are devoted to instruction in the Basic language. Simple programs are assigned. Students are expected to know enough Basic to write a short instructional program. The course simply introduces the language to the students so that they can begin to understand the microcomputer and analyze what is happening when they read a program. There are other courses offered at the University which teach not only Basic, but also the strategies involved in designing instructional computer programs (using the DEC-10). Those students who already have Basic language skills are excused from these lectures.

However, they work on compiling computer-literacy materials during the Basic session.

3. Systematic approach to program design and computer-based instructional strategies: Because students must write their own instructional programs on the microcomputer, they must understand that learning to program in Basic is only a small part of the instructional design process. A systematic approach to designing a computer program will help assure a high quality product. The following steps are discussed: writing objectives; developing an instructional sequence; construction and debugging the program; pilot testing and revision of the program; and final evaluation of the materials. Again, before students can develop their own programs or even select commercial ones, they must know the kinds of instructional strategies used successfully with a computer. Students examine various computer programs which represent drill, problem-solving, simulations, tutorial, testing, and computer management. In addition, the instructional strategies are related to specific computer capabilities such as randomization, immediate feedback and reinforcement.
4. Hardware and software selection: Three weeks are devoted to this topic. The students explore various hardware and software available and develop criteria for selecting each. These criteria are gathered through readings and reports. The class then develops a standard instrument for evaluation. The instrumemt is used to rate five hardware systems and two software programs. During the third week, a hardware and a software dealer are invited to demonstrate their products. Students should know enough at this point to question the dealers intelligently. Student response to having dealers come is not always positive. Often they feel the dealer is giving them a hard sell, but even so the experience is good preparation.
5. Proposal writing and funding sources: Students receive tips on how to write an effective proposal and how to review state, federal, and private funding sources which are available to educators for the purpose of starting school programs using microcomputers. Students are given the Apple Foundation Guidelines on which they may model their proposal assignment.
6. Implementing innovations or strategies for putting microcomputers in the classroom: Students are given practical suggestions as to how to plan for the implementation of microcomputers in the classroom. Case studies are reviewed. The Concerns Based Adoption Model is discussed, particularly in relationship to the diagnostic measurements used by the model (Scale of Concern and Levels of Use).
7. Panel discussion: "The Real World": Teachers representing elementary and secondary school levels discuss their experiences using microcomputers with children of all ages and at all levels. It is important for students to hear both positive and negative experiences which the panel members have encountered in their attempts to integrate the microcomputer into their course. This session brings a certain reality to the subject: students begin to realize that not all their pupils will respond well to the microcomputer and that not all principals are enlightened about the microcomputer's usefulness.
8. Graphics: One of the best features of the microcomputer is its ability to do graphics. This session is dedicated to programming graphics on the TRS-80 and the Apple. Students are given graphic programming assignments.
9. Future technology: combining video and the microcomputer and information networking: Students are given a demonstration of a system which allows the microcomputer to control a video cassette player so that the student can view a cassette and interact with the computer. Videodisc technology is also discussed, as is the potential for using the microcomputer as a terminal in a networking situation. Information networking systems such as the "Source" are discussed.
10. Resources for teaching computer literacy: During this session, students share the materials they have been compiling throughout the semester. The materials are print and non-print resources which would be helpful in teaching computer literacy. Such materials include texts, workbooks, software packages, filmstrips, films, slide tapes, kits, etc. The materials are presented with a written review so that an annotated bibliography can be compiled.

COURSE EVALUATION

At the time this paper was written, only the summer course evaluation was complete. Students were questioned in regard to the adequacy of the course format and content; the appropriateness of the readings; the difficulty of the assignments; desired course changes; what they liked best and least about the course; and whether course content would be best covered in a sixteen week course. The results are as follows.

1. Course format: Students preferred the course format which combined lecture, demonstration, hands-on experience, and outside resource people. They enjoyed variety. Students in the summer session felt that more hands-on experience would be beneficial
2. Course content: Students felt that the course content was more than adequate. There were suggestions that the topics of programming, instructional systems design, and methods of teaching computer-literacy should be covered in more depth.
3. Readings: Students were given a bibliography which contained readings in the categories of history, philosphy and research of computers; computer literacy and implementation; resources; and hardware, software, and programming. The bibliography was designed to aid students in their proposal writing. The summer students felt that the readings were helpful.
4. Assignments: In the summer session students had to do a simple CAI program and write a proposal. Students found the programming assignment enjoyable, however, the more experienced programmers felt that teaching Basic should be eliminated or a pre-requisite for the course. Other students suggested that students be grouped according to level of experience. Most of the students saw the benefit in writing the proposal, however, few liked the task. They felt it was too difficult, especially, in a six week time frame. Suggestions were to provide other options such as a review of literature or the development of a computer-literacy curriculum. These are now encorporated as options to the proposal.
5. Best and least: The questionnaire indicated that the students like the variety of presentation styles used in the course They enjoyed the hands-on experience and the opportunity to evaluate hardware systems. They disliked the fact that the six week time factor left them hard pressed to complete the assignments. In addition, they disliked the sales representatives from various microcomputer companies because the sales people were biased and sometimes poorly prepared.
6. Changes: The students recommended the following changes: a decrease in enrollment from thirty-five; elimination of the proposal; more software demonstrations; more hands-on experience; specific programming assignments; development of a software-sharing network; and more programming.

Not all of the recommendations made by the students were incorporated in the second course offering. The initial intention of the course remains the same. It serves as an introduction to the microcomputer and its use in education. The course objectives have been more clearly defined and the class assignments have been varied.

Diversity among students remains a problem. Although the majority of the students are at a low entry level, students who are experienced with computers and programming continue to sign up for the course. They are often frustrated by the basic thrust of the course. Perhaps a multi-track approach is the answer. This approach will take careful planning to implement.

THE FUTURE

Thus far, the course "Microcomputers in Education" has been very successful. Because the Media Education program is on the masters and doctoral level, the course is only offered to graduate students. There is a need to reach the undergraduate student. Presently, the College of Education Learning Resources Center is offering an orientation about microcomputers to undergraduate students enrolled in teaching methods courses. In addition, the Media Education department has included a microcomputer unit in their basic media course. However, this limited exposure is not enough. Plans are being made to offer an undergraduate course in computer literacy by spring of 1982.

It is not enough to just educate future and current teachers; faculty who are teacher educators need to be aware of the importance of developing computer-literacy skills for elementary and secondary school teachers and administrators. During the 1980-1981 school year, the College of Education Continuing Education Department is offering a series of workshops on microcomputer hardware, software, and computer literacy. It is the intention of these workshops to develop a computer awareness among faculty and school personnel. Of course, this is only the beginning. Extensive continuing education is needed to develop computing competencies among those involved in teacher education, but they must first be made aware of the need.

It may seem strange that the area of media education would be the one area of teacher education concerned with developing computer-literacy skills. But it is not difficult to understand in view of the history of the field. The field's major concern has been and still is the systematic selection and integration of media or technology in teaching. For years, media educators have been teaching teachers to use everything from the blackboard to the 16mm projector. Because the computer is a new technology which needs to be integrated into the classroom, it is only natural that media education have a hand in training teachers the computer's use in education. Computers, just like slide tapes or films, must be viewed by classroom teachers on all levels as a tool, one which takes careful thought and training to use effectively. The course "Microcomputers in Education" is only a beginning in an effort to meet this need.

REFERENCES

Dennis, J. "Stages of Development in Introducing Computing to Teachers." National Education Computing Conference Proceedings, Iowa City 1979.

Manji, Jamal F. "Birth of the Microcomputer Age in the Classroom." School Product News, August 1980.

Milner, S. "Analysis of Computer Education Needs for K-12 Teachers." National Education Computing Conference Proceedings, Iowa City 1979.

Moursund, D. "ACM Elementary and Secondary Schools Subcommittee Progress Report." National Education Computing Conference Proceedings, Norfolk 1980.

Taylor, R., Poirot, J., Powell, J. and Hamblen, J. "Computing Competencies for School Teachers: A Preliminary Projection for All but the Teachers of Computing." National Education Computing Conference Proceedings, Iowa City 1979.

Taylor, R., Poirot, J., and Powell, J. "Computing Competencies for School Teachers." National Education Computing Conference Proceedings, Norfolk 1980.

MICROCOMPUTER CLASSROOM USE PATTERNS

Dorothy H. Judd
Northern Illinois University

There have been a number of questions raised about the actual patterns of curricular use of microcomputers by teachers in K-12 schools. In the fall of 1980 an exploratory survey of selected Illinois schools was undertaken to determine the use patterns of teachers known to have microcomputers available in their classrooms. (The source for these respondents is reported in a note at the conclusion of this paper.) The questionnaire achieved a 64.5% response rate reresenting 127 respondents. Of these 127, eleven questionnaires were not used in determining the findings due to one defect or another. This report is based on the 116 respondents whose questionnaires were found useable.

The school district sizes reorted in Table 1 do not reflect any attempt to mirror the distribution of sizes in the school systems of Illinois. But every size district is represented in the responses reported in this study.

Table No. 1

Question: Size of school district

Answer Choice N=116	Number	Percent
1. under 999	21	18.1 %
2. 1000-3999	43	37.1
3. 4000-6999	18	15.5
4. 7000-9999	12	10.3
5. over 10,000	14	12.1
no answer	8	6.9
Total	116	100.0 %

The students' grade level is shown in Table 2. Reports have indicated that use of microcomputers is a junior high school and high school phenomenon, but grades K-3 and 4-6 show surprisingly high rates of use.

Table No. 2

Question: Grade level of students you teach using microcomputers?

Answer Choice N=116	Number	Percent
1. K-3	14	8.0 %
2. 4-6	35	20.1
3. 7-8	35	20.1
4. 9-10	41	23.6
5. 11-12	49	28.2
Total (multiple ans.)	174	100.0 %

The information in Table 3 holds little surprise. Students in regular classes dominate the type of students taught by the 116 teachers who responded to this survey. The number of teachers involved with gifted students is less than reports in the literature would have led one to expect. The extent of use in bilingual and in special education calls for further exploration.

The top eight uses of microcomputers in these classrooms in decreasing order are:

1. teach microcomputer operations
2. teach programming
3. computer literacy
4. teach computer role in society

5. teach use in general problem solving
6. use as a tutor (teach content/skills)
7. drill in math, spelling, etc.
8. run simulations of science/social events

Table No. 3

Question: Type of student you teach using microcomputers?

(multiple answers permitted)

Answer Choice N=116	Number	Percent
1. Regular	96	57.8 %
2. Bilingual	13	7.8
3. Special Education	25	15.1
4. Gifted	22	13.3
5. Other (what?)	10	6.0
Total	166	100.0 %

As Table 4 makes clear, "to teach data processing" would rank ninth, a full seven percentage points behind the "run simulation" answer.

When the answers teachers gave about their future plans for using microcomputers are combined with the answers given for patterns of present classroom use some changes occur:

> Computer literacy drops to fifth position, from third, and is replaced by "use as a tutor", which ranks sixth in current use pattern. Quite possibly this change reflects a growing anticipation of using authoring systems or authoring languages to create computer-assisted instructional (CAI) units to provide tutor services.
>
> Drill as a microcomputer activity jumps ahead in Table 5 to fourth from seventh.
>
> Along with the change from third to fifth of the "computer literacy" answer, "teach computer role in society" drops from fourth to seventh.

It is interesting that the first two positions in teacher ranking of present use and future use remain the same, "teach microcomputer operations" and "teach programming", while the eighth ranked choice, "run simulations of science/social events" remains eighth in both Table 4 and Table 5.

Besides asking how the microcomputers were being used, the current study asked what is the most and least curricular time any student had used a microcomputer in the four weeks preceding the receipt of the questionnaire. The detailed answers are found in Tables 6 and 7. Highlights of the findings are:

> 40.5% of the teachers reported more than 20 minutes use of a microcomputer per class period or clock hour as the most use by a student;
> 25.0% reported less than 20 minutes per class period or clock hour as the most use by a student, while;
> 22.4% reported no use at all, as the most use.
> 64.7% reported no use at all as the least use; while,
> 19.8% reported less than 20 minutes as the least use.

When these teachers were asked about their plans for use of the microcomputer for the remainder of the school year, they responded:

> 27.6% had no use plans.
>
> 27.6% planned to use microcomputers less than 20 minutes per class period.
>
> 22.5% planned to use microcomputers more than 20 minutes per class period.

A sizeable number reported that "time will vary" in their plans for the remainder of the year. See Table 8 for detailed responses. The question asking the respondents about their own personal use of the microcomputer revealed little except that their plans for student use and their own personal use tended to have the same time distribution.

Table No. 4

Question: To indicate the kind of microcomputer activity you use with your students: a) circle YES or NO

Answer Choice N=116	# YES	% YES
1. Use as a tutor (teach content/ skills)	60	51.7
2. Drill in math, spelling, etc.	58	50.0
3. Teach micro-computer operations	87	75.0
4. Teach programming	84	72.4
5. Teach computer role in society	62	53.4
6. Computer literacy	68	58.6
7. Teach data processing	30	25.9
8. Teach use in general problem solving	61	52.8
9. Run simulations of science/social events	37	32.9
10. Teacher generation of tests	21	18.1
11. Teacher generation of work-sheets	16	13.8
12. Teacher test scoring	14	12.1
13. Use in developing objectives	7	6.0
14. Use in developing instructional material inventory	17	14.7
15. Use in developing evaluation criteria	10	8.6
16. Other teacher uses (what?)	29	25.0

Table No. 5

Question: To indicate the kind of microcomputer activity you use with your students: c) indicate your future plans.

Answer Choice N=116	Future
1. Use as a tutor (teach content/ skills)	27
2. Drill in math, spelling, etc.	29
3. Teach micro-computer operations	15
4. Teach programming	15
5. Teach computer role in society	7
6. Computer literacy	17
7. Teach data processing	3
8. Teach use in general problem solving	10
9. Run simulations of science/social events	9
10. Teacher generation of tests	9
11. Teacher generation of work-sheets	9
12. Teacher test scoring	12
13. Use in developing objectives	6
14. Use in developing instructional material inventory	7
15. Use in developing evaluation criteria	2
16. Other teacher uses (what?)	1

Table No. 6

Question: In the past four weeks what has been the most curricular time use by any student in your classes?

Answer Choice N=116	Number	Percent
1. no use	26	22.4 %
2. 10 minutes or less per class period/clock hour	17	14.7
3. 11-20 minutes per class period/ clock hour	12	10.3
4. 21-30 minutes per class period/ clock hour	16	13.8
5. over 30 minutes per class period/ clock hour	31	26.7
6. time varies	8	6.9
no answer	6	5.2
Total	116	100.0 %

Table No. 7

Question: In the past four weeks what has been the least curricular time use by any student in your classes?

Answer Choice N=116	Number	Percent
1. no use	75	64.7 %
2. 10 minutes or less per class period/clock hour	18	15.5
3. 11-20 minutes per class period/ clock hour	5	4.3
4. 21-30 minutes per class period/ clock hour	1	0.9
5. over 30 minutes class period/ clock hour	3	2.6
6. time varies	6	5.2
no answer	8	6.9
Total	116	100.0 %

Table No. 8

Question: What is your plan for microcomputer curricular use by your students in the remainder of the school year?

Answer Choice N=116	Number	Percent
1. no use	32	27.6 %
2. 10 minutes or less per class period/clock hour	21	18.1
3. 11-20 minutes per class period/ clock hour	11	9.5
4. 21-30 minutes per class period/ clock hour	9	7.8
5. over 30 minutes per class period/ clock hour	17	14.7
6. time varies	20	17.2
no answer	6	5.2
Total	116	100.0 %

Every survey effort faces its moment of truth: the questions that should have been asked, but were not; the questions that might have been phrased differently, but were answered anayway. This study revealed a good deal less microcomputer time on task than might represent a good return on the investment. The candor of the respondents is refreshing. When they are not planning on using a microcomputer, they have said so. They indicated plans for quite fundamental microcomputer activities, although they could have claimed more sophisticated endeavors. The strength of any exploratory study is found in determining the status of the subject inquired about. In that sense, the generous response and helpful comments of the 116 respondents to the present survey have made it a success.

NOTE:

The questionnaire was mailed October 18, 1980, which was selected to avoid opening school year activities. It was mailed to 197 teachers whose names were listed as members of user groups in Illinois or who were listed in either "A Microcomputer Registry" created by Dr. Gary Tubb of Illinois State University or in a report entitled "Extent of Computer and Microcomputer Use in Schools and Colleges in Northern Illinois" authored by Richard Gaston of Governors State University. A follow-up mailing was made on November 10, 1980, to encourage the return of additional questionnaires. Questionnaires returned prior to the follow-up mailing numbered 102; the follow-up mailing brought an additional 25, for a total return of 127 of the 197 questionnaires mailed, or a 64.5% return. The questionnaire was printed in booklet format, and the first page was a cover letter explaining the purpose of the study. Respondents were encouraged to express additional comments or opinions. Each questionnaire provided space for name and address to permit tracking of respondents who did not return their questionnaire. Each questionnaire booklet provided for easy return mailing. The respondents simply folded and stapled the booklet so that the return address and stamp were in place.

EDUCATING URBAN ELEMENTARY TEACHERS IN COMPUTER SCIENCE

Tim Carroll and Nancy Johnson
Department of Mathematics
Chicago State University
Chicago, Illinois 60628

INTRODUCTION

During the summer and fall 1980, we taught a sequence of two courses in computer science for elementary teachers. The courses were offered under an NSF grant for the continuing education of elementary teachers.[1] The grant writer and project director was Ramona G. Choos, Associate Professor of Mathematics at Chicago State University. The target group for the project was elementary school teachers (grades 6-8) from Chicago public schools.

The two classes were "Introduction to Computer Science & Computer Literacy for Elementary Teachers," offered during July 1980, for sixteen 3-hour sessions (4 per week), and "Computer Science for Teachers," offered during fall term 1980, for sixteen 3-hour sessions (1 per week). The summer course (enrollment 14) was taught by Nancy Johnson, and the fall course (enrollment 30) was taught by Tim Carroll.

The goals of the courses were to acquaint the teachers with uses of the computer, computer terminology, programming, and curriculum activities. Since the courses were intended to be separate, the first was not a prerequisite for the second. We tried to keep overlap of topics to a minimum, yet make each accessible to a beginner.

A variety of topics were introduced in the "Computer Literacy" course. But the main topics of the other course were programming and computer applications relevant to the classroom. Since the sessions were each three hours long, we decided that the best approach in both courses would be to talk for an hour or so and use the remainder of the time working in the lab. The lab in this case was the terminal room which contained approximately fifteen terminals giving access to a CDC Cyber 170-730. The

---

[1] NSF Grant SPI - 8001212.

terminals were of a wide variety including both CRTs and hardcopy terminals. Although it was impossible to reserve the lab for our classes, we used it at non-peak hours so there were always terminals available. (In the summer the lab was virtually empty. However, in the fall we encountered difficulties because the lab was used more heavily.) We also had a mathematics laboratory available with a small, but growing collection of computer books and activities for use in the elementary school.

"Computer Literacy for Elementary Teachers": In this class students were introduced to a variety of topics. As has been already mentioned, the first half of each session was used as a lecture and the second half was lab. The two parts of a session were not necessarily coordinated, because many topics introduced didn't have a related computer activity.

The text, Marilyn Bohl's Information Processing, 3rd edition [2], was chosen to provide a general over-view of computers. In addition, the text contained a glossary of computer terminology which the students could use for reference.

The following is a list of some of the topics covered. These did not necessarily have a lab counterpart.

1) "What is Computer Literacy?"
2) History of computers
3) Uses of the computer
4) Employment opportunities
5) Parts of the computer
6) Computer hardware
7) Computer literature

The materials used for exploring the first topic were articles by Taylor, Poirot, Powell [9], Johnson, Anderson, Hanson, Klassen [7], and "An Agenda for Action" [1]. Only the first two chapters of the text were used. (We also used the Instructor's Guide [3] and the Study Guide [4].) There seemed to be little time to return to the text after the students became experienced in the computer. For an introduction to CAI, we used the article by Chambers and Sprecher [5]. At least two lecture sessions were scheduled in the math lab to allow the students time to examine various books, journals, and magazines, and to instruct the class how to use the paper computer [6].

The topics for the lab portion were:

1) game playing;
2) information and instructional uses of the computer; and
3) introduction to Basic programming.

The students were introduced to computers by playing games. A library of approximately 200 games was available (not all of them very good). After two sessions they were able to "log-on." obtain the game library, choose a game, and play it.

Next, the students were introduced to other uses of the computer through a mathematical program and the "Help" system.

After approximately six sessions of using the terminals, the class was relatively comfortable with the computer. Each of them had used dial up, non-dial up, printing, and non-printing terminals. At this point Basic was introduced. The whole-program approach was used. The students were first given an example of a complete terminal session for entering, listing, and running two simple Basic programs. The programs were explained and the students were required to follow the example, and then modify the two programs.

The Basic statements taught were: REM, LET, PRINT, INPUT, IF, GOTO, END. Since the whole-program approach was used and there was no text for Basic, the students were left to discover syntax rules on their own. The programming structures taught were straight-line, double-alternative decision, and loops. The techniques of counting and summing were presented. In addition, creating self-documenting programs was stressed.

The students were expected to complete several computer assignments, including written evaluations of games they had played, as well as writing simple programs. A term project was also required. Students were allowed to write a paper or do a programming project. Among the projects were: a microcomputer grant proposal, a paper on medical uses of the computer, a program to teach modular arithmetic, and a program to keep class attendance records for a school. The last six sessions were largely occupied with helping students prepare their projects. During the last two sessions the students presented their projects to the class.

"Computer Science for Elementary Teachers": In the fall we continued our program for the elementary school teachers with a sixteen-week course. The class met once each week for three hours in the evening. Each of these class periods was divided into two parts. The first part, lasting one to one and a half hours, was in a usual classroom setting and consisted of lecture and classroom discussion. The second half of the class was spent in the computer lab.

The text, Kemeny and Kurtz's Basic Programming, 3rd edition [8], was chosen because it contains many different application programs.

The principal objectives were to be:

1) able to read and write Basic language programs.
2) exposed to and be able to use educational software.
3) conversant with computer terminology.
4) able to discuss problems that lend themselves easily to computer solution and those that do not.

We did not use the computer for the first class, but spent the entire period discussing general back-ground material including flowcharting, computer languages, computer programs, and algorithms. After that, a typical class consisted of resolving problems that they were having, a short lecture on new material, and a discussion of handouts that exemplified the lecture. Their assignment was to either modify a handout program, or create a program that used the information given. Our goal was not to make computer programmers out of these teachers, but rather to expose them to computer programming and show how they could use existing or modified programs for their classrooms. For example, we created record keeping, grade averaging, and social science drill and practice programs, as well as programs in math and physical science. By the end of the semester, most students were able to use looping techniques (including nested loops), one- and two-dimensional arrays, functions (built-in and defined), subroutines, string variables, and random numbers.

In addition to teaching a programming language we also discussed the computer and how it could be used in their classrooms. Since most of the teachers were from the nearly bankrupt Chicago public school system, they were quite pessimistic about the chances of their classrooms ever having access to a computer. The newspaper article "Homework On Computer Becomes a 'Class' Distinction" [10] stirred some interest in obtaining funds either through grants or independent fund sources.

At the middle of the semester we brought in an Apple microcomputer. We demonstrated what this little computer could do, comparing and contrasting this with our interactive time-sharing system. Since there was only one machine and thirty students, little hands-on time was afforded. Their assignment for this class was to research the microcomputer field. The purpose was not only to gain hands-on experience with a micro and its graphics capabilities, but also to acquaint them with the various educational software available. (Their final projects reflected this. One of the teachers wanted to write a tic-tac-toe program similar to a micro program she had seen, but found that she was unable to do so. However, her final product showed added touches gained from this experience.) They were to visit at least three microcomputer stores, write up the comparisons, and give a short oral presentation to the rest of the class. The advantage of residing in a metropolis such as Chicago is access to a vast number of suppliers of hardware and software, including publishing companies. In order to communicate successfully with manufacturing representatives, our students had to quickly become conversant with computer jargon. Although the class, as a whole, was very apprehensive about talking to salespersons, they found this experience to be very interesting and fun as well as informative. Instead of complaining about teaching in a school district on the verge of bankruptcy, they were talking about getting a relatively inexpensive microcomputer in their classroom.

As a final project, each person in the class was to write a program of personal interest. For the most part, they wrote programs that could be used in their classrooms: either drill and practice programs, games, or class management programs. The most ambitious was an interactive program asking students questions about themselves, their teacher and their school. After his program was completed, one of the teachers brought his own class to the computer lab to run his project as well as other programs and games. Other final projects were programs that found the gcd and lcm of any two integers; identified nouns, verbs and adjectives; translated Roman numerals; and computed angles of triangles; and developed class lists from files of SAT scores. There were also drill and practice programs in mathematics and finite field arithmetic, and number guessing and word scrambler games.

Evaluation: Because the main thrust of the summer course was computer literacy, we devised a self-evaluation for the students to take on the first day of class and again on the last day of class. The form was based on the competencies Cl.1-Cl.7 appearing in "Computing Competencies for School Teachers," [9], and here briefly summarized:

Cl.1 read and write simple programs
Cl.2 have experience using educational software
Cl.3 knowledge of computer terminology
Cl.4 knowledge of types of problems that can and cannot be solved by computers
Cl.5 use educational computing information
Cl.6 discuss history of computing
Cl.7 be able to discuss moral issues related to societal uses of the computer.

Students rated themselves on a scale of 1 (not competent) to 10 (competent). The results of the self-evaluation are given here; the entries represent the mean response:

	First day	Last day
Cl.1	1.8	7.1
Cl.2	1.4	6.9
Cl.3	3.2	7.3
Cl.4	1.5	5.9
Cl.5	1.7	7.1
Cl.6	1.7	7.1
Cl.7	1.8	7.4

The relatively high rating for Cl.3 for the first day might be explained by the use of computer jargon in the newspapers and on TV. Competency Cl.4 was not stressed in class. Finally, Cl.7 was not touched on at all in class, yet the class felt as competent in that as in, say Cl.1, on which we spent at least one-fourth of class time.

Judging by these self-evaluations and other written comments, the class felt confident of their ability to learn about and to use computers. The best part of the course, both for the students and the teacher, was the lab. Because the class met nearly every day, they remained interested and enthusiastic and didn't forget things from one session to the next.

Some of the programming projects they did were impressive. Some had to learn a great deal more Basic than was presented in class: FOR/NEXT loops, arrays, and generating random numbers. Some of those who wrote papers were envious of those who wrote programs because the programmers were having so much fun working on their projects!

As mentioned before, the fall class was composed of students from the summer class as well as new students. For the first few weeks of the semester the new students seemed to be quite apprehensive about the material they would be required to know. The experience from the summer made those students feel confident, and this self-assurance was felt strongly by the new students. Gradually the apprehension melted as the newer students gained confidence from working on the computer.

Because of mechanical breakdown and other students, four or five terminals often had to suffice for our class of thirty. This was not acceptable and was commented on most often. As a whole, however, the class was very enthusiastic about the program and when asked if they would be interested in taking more courses, all of the respondents said "yes."

One suggestion was given that we conduct a microcomputer class. A room with fifteen to thirty microcomputers available the entire period would certainly alleviate propblems of competition. Since the hands-on time was, as in the summer, the part of the class most enjoyed, the intensive use of microcomputers seems to us to be a direction to aim future classes.

REFERENCES

1. An Agenda for Action, National Council of Teachers of Mathematics, 1980.

2. Bohl, M., *Information Processing, 3rd edition*, SRA, Chicago, 1980.

3. Bohl, M., *Instructor's Guide: Information Processing, 3rd edition*, SRA, Chicago, 1980.

4. Bohl, M., *Study Guide: Information Processing, 3rd edition*, SRA,

5. Chamber, A. and Sprecher, J., Computer Assisted Instruction: Current Trends and Critical Issues, Communications of the ACM, Vol. 23, no. 6, pp. 332-342 (1980).

6. Instructo Paper Computer, McGraw-Hill, 1979.

7. Johnson, D., Anderson, R., Hansen, T. and Klasse, D., Computer Literacy -- What is it? The Mathematics Teacher, Vol. 73, no. 2, pp. 91-96 (1980).

8. Kemeny, J. and Kurtz, T., *BASIC Programming, 3rd edition*, John Wiley and Sons, New York, 1980.

9. Taylor, R., Poirot, J. and Powell, J., Computing Competencies for School Teachers, Proceedings of NECC/2, pp. 130-136 (1980).

10. Walker, R., "Homework on computer becomes a 'class' distinction", The Christian Science Monitor, June 6, 1980, p. 1.

MICROCOMPUTER NETWORKS

Robert F. Tinker
Richard E. Pogue

ABSTRACT: Novel Microcomputer Configurations--Implications for Teaching

Robert F. Tinker, Technical Education Research Centers, 8 Eliot Street, Cambridge, MA 02138

The Technology Center at TERC has been experimenting for several years with ways of using microcomputer technology to aid teaching. In this presentation, the results of our experience with high resolution graphics, computer-based instrumentation, and local networks will be reported.

Abstract: A Strategy for Integrating Microcomputers Into An Existing Statewide Academic Computing Network

Richard E. Pogue, Health Systems & Information Sciences, Medical College of Georgia, Augusta, GA 30901

The University System of Georgia Computer Network (USCN) was established in the early 1970s to provide academic computing services to faculty and students at the more than thirty state-supported institutions of higher education. As a major user of the USCN, the Medical College of Georgia (MCG) decided in 1978 to investigate the potential of microcomputers for use in conjunction with the powerful computing system provided by the USCN. A plan to use communicating microcomputer stations for both stand-alone use and as the primary means of access to the USCN has been underway for almost two years, and has been accepted enthusiastically by the MCG community.

The conference discussion will include initial implementation strategies, guidelines for selecting and incorporating microcomputers into each academic computing application area, recommendations concerning network support for microcomputers, and plans to manage growth of microcomputers as a major network computing resource.

ADA TUTORIAL

Robert F. Mathis
Old Dominion University
Norfolk, VA 23508

ABSTRACT:

Ada is a new programming language developed by the Department of Defense for use in embedded computer systems, but its area of application is much wider. There are already indications that it may replace Fortran, Basic, Pascal, and even Cobol in many situations. This tutorial reviews the Ada programming language and its relevance to teaching computer programming and the use of computers in education.

The first session will be a general introduction to Ada. Although the emphasis will be on how the language might be used by people with small systems and a background in only Basic or Fortran (people with a Pascal background would be very well prepared for this session), almost the entire language will be reviewed. This session will review the development of the language and its relevance to programming in the 1980s.

The second more advanced session will deal primarily with some larger programs using case studies and emphasizing top-down development, modularity, new types, encapsulation, and concurrent programming. The primary focus of attention during the second session will be an intelligent terminal program for use on a machine like the Terak 8510/a. People intending to come to the second session should attend the first one also for continuity.

COLLEGE COMPUTER LITERACY

Edward B. Wright
T. P. Kehler
M. Barnes
James W. Garson
David Miles

ABSTRACT: Infusion of Computer Science Into Liberal Arts and Teacher Education Curriculum

Edward B. Wright, Academic Computing,
Oregon College of Education,
Monmouth, OR 97361

Oregon College of Education began a program starting spring 1980 to help each department use microcomputers as an integral tool within their offerings. This is being accomplished by:

1. A faculty training program which allows a minimum of ten instructors each year to develop a course module.
2. Setting up and equipping two instructional labs with microcomputers and peripheral equipment. (A total of 15 Apple II microcomputers have been purchased to date.)
3. Developing and/or modifying software to meet course needs.

During the academic year 1980-81, professors from music, mathematical writing, Spanish, secondary education, and psychology have been using microcomputers in their classes. A report of these activities will be available.

ABSTRACT: Design of Interactive Help Systems

T. P. Kehler and M. Barnes,
P. O. Box 225936, MS 371, Dallas,
TX 75265

Help systems generally do not take advantage of the great number of on-line documents usually found on most time-sharing systems. This presentation will focus on research done on the HELPME Help System, developed by the authors, which attempts to deal with this issue.

HELPME is a Lisp-based system designed to provide on-line help for novice and expert users of computer systems. HELPME permits implementation of easy to use interfaces to existing documents by allowing a user familiar with a document (a "document expert") to produce an index and incorporate information relating to the structure of the document into the interface. A typical user of HELPME can then interact with the document and index through a series of commands to quickly find the information desired.

The presentation will focus on the current state of HELPME and what has been learned during its development. Key problems, such as how maintenance of documents affects indexes, will be briefly presented to determine how realistic a system such as HELPME is for time-sharing environments.

ABSTRACT: Interactive Computer Graphics for Computer Literacy

James W. Garson, Department of Information Engineering, University of Illinois at Chicago Circle, Box 4348, Chicago, IL 60680

A series of four programs, written for the Apple II in Pascal, that display the use of computer graphics in a well-defined educational role, will be presented. The programs in our series help the user learn basic concepts concerning the architecture, or functional structure, of hand calculators.

The first program in our series displays the architecture of a very simple four-function calculator. The second is slightly more complex because its calculator has an extra register for an automatic constant. The third program presents stack architecture while explaining stack parsing, and the final program displays a simple programmable calculator.

A number of issues will be discussed concerning the way the user interacts with these programs, their graphics design, and their educational philosophy.

ABSTRACT: Teaching Watfiv on an IBM 370 with Micro-assistance

David Miles, Humanities Department,
Ohio University Belmont Campus,
St. Clairsville, OH 43950

The Problem: I teach at a regional campus of Ohio University, 135 miles from the main campus. The campus is too small to have its own computer science staff or hardware, and there is no college-wide network for data communications. A new university general education requirement includes a course in "quantative skills." A course in programming is one of prime interest to students, but can we provide our students the opportunity of this course?

The programming course in the university curriculum is a course in Watfiv. Attempts to teach the course locally through a nearby technical college had not always been entirely satisfactory, largely because of the business orientation of their staff, curriculum, and hardware.

Our Solution: Two years ago we acquired a small lab of microcomputers (TRS-80) to use in a computer-literacy program. We decided to teach the material in the following manner:

1. Students are taught programming concepts on microcomputers using an enhanced Basic interpreter which includes structured programming elements.

2. Students translate their enhanced Basic programs into Watfiv programs.

3. By using a word processor, students create a cassette tape text file of their Watfiv programs.

4. Students load their tape into an intelligent microcomputer functioning as a terminal to expedite delivery and execution of their program by dedicated phone line on the remote IBM 370 system.

APPLICATIONS OF THE DEFINITE INTEGRAL AND BASIC

Dr. Peter A. Lindstrom
Genesee Community
College
Batavia, NY 14020

SECTION 1 INTRODUCTION

During the Fall 1980 semester, my teaching schedule included the only section of Calculus II offered at Genesee Community College. Having already taught this course many times, I decided to introduce some new teaching techniques and some new course objectives to stimulate the interests and needs of the students. Some of my new course objectives were to:

1) present meaningful applications of the definite integral to solving real-life problems in Calculus II, as so often applications presented in texts do not illustrate real-life situations;

2) use the Genesee Community College Prime 300 computer as a computational tool to help solve these applied problems as often as possible;

3) use the Genesee Community College Prime 300 computer to present new teaching techniques to enhance the topics taught in the course; and

4) use the computer to teach the Calculus II students some of the fundamentals of Basic, but not in a formal course setting.

This paper shows how these objectives were fulfilled and the consequences of meeting them. Section 2 presents a brief history and summary of UMAP modules, two of which were used to help achieve these objectives. Section 3 presents a short history and philosophy of computers at Genesee Community College. Section 4 tells how these objectives were met in this course and the consequences that followed. Section 5 summarizes my experiences in fulfilling these objectives.

SECTION 2 UMAP MODULES

In the 1970s, the number of students who needed to learn mathematics, mathematics modeling, and applications of mathematics for their professional careers grew rapidly. During these same years, teachers and people in business and government who use mathematics daily realized that there was a great need for new instructional materials to teach mathematics and show how mathematics can be applied to solve problems in non-mathematical areas. In July 1976, the National Science Foundation funded the Undergraduate Mathematics and Its Applications Project (UMAP) to the Education Development Center, Inc. (EDC) of Newton, MA. Two primary purposes of UMAP are:

1) to develop lesson-length modules from which students may learn professional applications of mathematics for solving problems in non-mathematical areas, and mathematical topics that are needed in the undergraduate classroom, and

2) to create a consortium of authors, reviewers, field-testers, users, and department representatives that would continue to produce and disseminate these modules after the funding has ended.

Presently, there are more than 350 modules in various stages of production and more than 2400 consortium members. Modules have been developed in political science, biology, the social sciences, the physical and environmental sciences, economics, and business. Areas of mathematics include calculus, finite math, statistics, linear algebra, differential equations, operations research, and precalculus mathematics. Each UMAP module is written for student use and requires no expertise by either the teacher or student beyond that described in the prerequisite list contained within the module. So, the student need not have any special training in mathematics modeling or special experience with the non-mathematical subject matter.

SECTION 3 HISTORY AND PHILOSOPHY OF COMPUTERS AT GENESEE COMMUNITY COLLEGE

Since Genesee Community College opened in September 1967, it has offered a two-year career program in data processing, along with related options. One of the program's strengths has been its emphasis on "hands-on experiences" in both programming and computer operations. The programming languages taught include Assembler, Cobol, Fortran, PL/1, and RPG. But the data processing program has had numerous weaknesses over the past years, primarily that there was no provision for using a terminal under the IBM/360 system that the college had been using.

In 1979, Genesee Community College applied for and received a Vocational Education Act (VEA) Grant to purchase a minicomputer system. In January 1980, the college installed a Prime 300 minicomputer with 128 kilobytes of main memory, five CRT terminals, one hardcopy terminal, and one 180 character per second line printer. With this new equipment, numerous changes have taken place: Basic and Pascal are now taught, the data processing students have experience using terminals, and the faculty and students from other areas of the college are using the computer.

SECTION 4 FULFILLMENT OF THE COURSE OBJECTIVES

A. Course Information:

The course carries four semester hours of credit. It requires Calculus I and used Calculus With Analytic Geometry, Second Edition, by Earl W. Swokowski as a text. There were 11 students enrolled. (This is a typical size for a Calculus II course for a Fall Semester; the Spring Semester averages about 25 students.) The student's majors were: Mathematics/Science (3), Computer Science (2), pre-engineering (3), and liberal arts (3). The student's computer background included knowledge of Fortran (1), Cobol (1), Fortran and Cobol (2), Basic (2), and no programming language (5).

B. UMAP Module #323:

The first topic was developing the definite integral as the limit of Riemann Sums. Instead of using the approach presented in the text, I selected UMAP Module #323, "Developing the Fundamental Theorem of Calculus," by Peter A. Lindstrom. This module develops the Fundamental Theorem of Calculus by first presenting applications of the limit of Riemann Sums to area, work, and distance problems, three areas rich in applications of the definite integral. The module emphasizes setting up the appropriate Riemann Sum that leads to a definite integral for solving these problems. Also, many properties of the definite integral are introduced by these applications. (See Appendix A for excerpts of UMAP Module #323.)

After spending four class periods (80 minutes each) on this topic, we then studied for two class periods the same topics from Chapter 5 of the text. The last section of Chapter 5, "Numerical Integration," was purposely not covered at this time. (The reason will be discussed later.)

C. Applications of the Definite Integral from the Text:

By the start of the third week, we were studying Chapter 6, "Applications of the Definite Integral, " from the text. The topics studied were area, volume, work, force exerted by a liquid, arc length, and applications to economics and biology. I have always felt that many of these applications are very superficial because

1) the students have not yet been exposed to enough of the techniques of integration so that the integrands have to be rather special ones, and

2) the students have not yet been exposed to the elementary transcendental functions, so many non-mathematical applications have to be avoided at this time.

D. Calculus II Project:

Upon completing Chapter 6 in the text at the end of the fifth week, the students were then assigned a project to read UMAP Module #379, "Elementary Techniques of Numerical Integration," by Wendell L. Motter, and do the exercises in the module. Upon completion of the module, the students were assigned problems from the text that were related to the module.

The purposes of the project were to:

1) have the students learn some numerical integration techniques. (This topic was purposely omitted from the text when we studied Chapter 5.) The module emphasized the Trapezoidal and Simpson's Rules;

2) have the students learn some of the fundamentals of Basic; (See Appendix B for excerpts from the module)

3) encourage the students to write their own programs and to use them with the exercises of the module and the text. Even though most of the students did not

know Basic, enough information is contained in the module for all of these students to learn many of the fundamentals of Basic.

E. The Prime 300 and the Project:

To assure that each Basic program of the module was compatible with the Prime 300, I tested each program before the project was assigned. In turn, a list of necessary program changes was developed so that the students were able to run the module programs and the ones they wrote.

Each student was assigned their own user ID and password.

The class spent a full classroom period becoming familiar with some of the operations of the Prime 300: how to LOGIN, LISt, FILE, etc. A list of such information was also given to each student.

F. Consequences and Conclusions of the Project:

The project and the Prime 300 prepared each student to do additional assignments for the course:

Assignment #1 (See Appendix C): As previously mentioned, after being introduced to the Fundamental Theorem of Calculus and some of the applications in the text presented in Chapter 6, the students see many superficial applications of the definite integral as they have not yet been exposed to enough of the techniques of integration. This assignment enabled the students to see other applications (not presented in Chapter 6). These applied problems were solved by numerical integration techniques on the Prime 300, so that the students did not have to determine the actual antiderivative of the integrand. When Chapter 10, "Additional Techniques and Applications of Integration," was studied, this assignment was then given again and the students then solved the same problems by the Fundamental Theorem of Calculus and the techniques of integration.

Assignment #2 (See Appendix D): As previously mentioned, after being introduced to the Fundamental Theorem of Calculus and some of the applications in the text presented in Chapter 6, the students see many superficial applications of the definite integral as they have not yet been exposed to the elementary transcendental functions. This assignment enabled the students to see other applications of the definite integral to non-mathematical areas. These applied problems were solved on the Prime 300. Later, they were able to solve the same problems by using the Fundamental Theorem of Calculus after more of the details of the elementary transcendental functions had been studied.

Assignment #3 (See Appendix E): One exercise of UMAP Module #379 (See Appendix B, Exercise 3) is to use the Trapezoidal Rule to approximate

$$\int_1^2 (1/x)\,dx \quad ,$$

which by definition = ln(2). After completing the project, we began our study of the natural log function, ln, by using

$$\ln(x) = \int_1^x (1/t)\,dt \ .$$

Just prior to this, Assignment #3 was given, allowing the students to use the computer to approximate ln(x) for various positive values of x. The assignment introduced the students to some of the properties of the ln function (e.g., $\ln(x\cdot y) = \ln(x) + \ln(y)$, etc.).

Assignment #4 (See Appendix F): So often, calculus students think of π as that number which is approximated by 3.14 or 22/7 or as that number that is the area of a unit circle. This assignment showed the students how π can be approximated by the computer, the programs they wrote, and two applications of the definite integral, area and arc length. This assignment was also given a second time after the students had studied the inverse trig functions (Chapter 9) and the trig substitution method for integration (Chapter 10).

Through the combined use of the two UMAP Modules, the Prime 300 minicomputer, and the additional assignments, I feel that the four objectives listed in Section 1 were fulfilled. Although I do not have statistical data or evidence to support these beliefs, the rather small class size (11 students) would not necessarily produce accurate conclusions. Considering each objective, I feel that the following conclusions are true:

1) UMAP Modules #323 and #379, the text, and additional assignments #1 and #2, presented a variety of interesting applications of the definite integral to both mathematical and non-mathematical areas. I have always felt that students appreciate mathematics more when they see answers to their questions, "Why do we have to study this?" and "How can we use this?"

2) The Prime 300, the Basic programs in UMAP Module #379, and the Basic programs written by the students gave them approximate results to many of the applied problems presented in the text and in the additional assignments.

3) The Prime 300, UMAP Module #379, the Basic programs written by the students, and additional assignments #3 and #4, presented topics to the students in ways that are different from the typical textbook approach. These teaching techniques worked well because the student is an active participant and not just a spectator in a classroom-lecture presentation.

4) Through the combined use of the computer, UMAP Module #379, the additional assignments, and the programs written by the students, these Calculus II students received a good introduction to some of the fundamentals of Basic. The two students who already knew Basic were able to immediately write the necessary programs; they could relate material from their Calculus II course to the previously studied computer course, and vice-versa. As for the other nine students, they grasped these fundamentals of Basic quite rapidly from the instructions presented in UMAP Module #379. Most of these students had many reservations about doing the project because they had no experience with Basic and no classroom instruction in Basic was going to be given. Such fears were short-lived once the students began working on the project, writing their own programs, doing the additional assignments, and seeing the results of their work.

SECTION 5 SUMMARY

Having taught mathematics at the college level since 1963, this was my first experience of combining Basic with a specific mathematics course. Although I do not have any data and statistical evidence to support my claims, I feel that the students were very pleased with these four objectives. The students became very interested in Basic and using the Prime 300, because many of them (who had no prior programming experience) enrolled in a computer science course the next semester, some asked for more assignments to be done on the Prime 300, some did some of the textbook homework assignments on the computer, and a few devoted too much time to the computer and did very little work in their calculus course.

Next semester (Spring 1981), I am teaching Calculus II again. I expect that 20-25 students will enroll in the course. For this class, I plan to include the four objectives previously discussed, teach the course in basically the same manner, assign the same project, and give the same additional assignments. I also plan to give more additional assignments that will combine as many of these four objectives as possible, along with encouraging these students to use Basic and the Prime 300 as often as possible.

APPENDIX A EXCERPTS OF UMAP MODULE #323

Output Skills:

1. To gain experience in computing limits of Riemann Sums.
2. To gain experience in computing definite integrals.
3. To gain experience in solving elementary area, distance, and work problems by integration.

Page 3:

Consider the three following problems and their solutions:

Example 4. Find the area of a right triangle whose base is 15 feet and whose altitude is 10 feet.

Example 5. Find the distance traveled by a car whose velocity at time t is v(t) = ((-2/3)t + 10) ft/sec when it travels from t = 0 to t = 15 seconds.

Example 6. Find the work done in lifting a bag of salt a distance of 15 feet above the ground, assuming that the bag has a hole in it so that at height d above the ground its weight (magnitude of downward force) is g(d) = ((-2/3)d + 10) lb.

Pages 7 and 8:

Consider then the product

(*) f((i)•(15/n))·(15/n),

which may be interpreted in the following three ways:

1. For Example 4, (*) is the area of the ith rectangle with width f((i)•(15/n)) and length (15/n), where the width is determined by the right endpoint, (i)•(15/n), of the ith subinterval.

2. For Example 5, (*) is the approximate distance traveled over the ith subinterval of time length (15/n) and where the velocity, f((i)•(15/n)), is constant over the interval and it is determined by the right endpoint, (i)•(15/n), of the ith subinterval.

3. For Example 6, (*) is the approximate work done over the ith subinterval of distance (15/n) and where the force, f((i)•(15/n)), is constant over the interval and it is determined by the right endpoint, (i)•(15/n), of the ith subinterval.

Pages 11 and 12:
Exercises

9. Set up and simplify a Riemann Sum for the function $f(x) = 2x + 4$ over $[1,8]$, using a partition with subintervals of equal length and selecting the left endpoint of each subinterval for c_i.

10. In Exercise 9 above, let n take on various positive integers values.

11. Interpret the numerical results of Exercises 9 and 10 above in terms of Examples 4, 5, and 6.

12. In the results of Exercise 9 above, let $n \to \infty$ to find the value of

$$\int_1^8 (2x + 4)dx.$$

Page 19:
Exercises

25. Find the area of the region bound by the x-axis and the graph of the parabola $y = 6 - x - x^2$.

26. Let $v(t) = t^2 - 3t + 2$ be the velocity of a car when $0<t<3$, where the velocity is measured in ft/sec and the time t is measured in terms of seconds. WARNING: What does a negative velocity indicate?

27. A bag of salt originally weighing 144 pounds is lifted upward. The salt leaks out uniformly at a rate so that half of the salt is lost when the bag has been lifted 18 feet. Find the work done in lifting the bag this distance.

APPENDIX B EXCERPTS OF UMAP MODULE #379

Suggested Support Material: Either a minicomputer or timesharing terminals which allow programming in Basic will be needed.

Prerequisite Skills:
1. Calculus through the definite integral and Riemann Sums.
2. The analytic definition of the integral.

Output Skills:
1. Write programs in Basic to calculate Riemann Sums for various functions.
2. Approximate integrals using the Trapezoidal Rule and Simpson's Rule and write programs in Basic to do this.
3. Explain why the Trapezoidal Rule and Simpson's Rule give better approximations of most integrals than rectangle approximations.

Page 6:
Example Program 1

```
100 DEF FNA(X) = 1/(x↑2 + 1)
110 INPUT A,B,N
120 LET D = (B - A)/N
130 LET S = 0
140 FOR X=A TO (A+(N-1)*D) STEP D
150 LET S = S + FNA(X)
160 NEXT X
170 PRINT S*D: 'IS THE LEFT-RECTAN.
        APPROX.'
180 END
```

Page 11:
Exercise

3. Compute $\int_1^2 (1/x)dx$ by hand calculations using the Trapezoidal Rule for N = 6. Note the exact value is ln(2).

Page 20:
Example Program 3

```
100 DEF FNA(X)=4*X↑3-3*X↑2+2*X-7
110 INPUT A,B,N
120 LET D = (B - A)/N
130 LET S = 0
140 FOR X=A+D TO (A+(N-1)*D) STEP 2*D
150 LET S=S+FNA(X-D)+4*FNA(X)+
           FNA(X+D)
160 NEXT X
170 PRINT S*D/3: 'IS SIMPSON'S APP.'
180 END
```

Page 20:
Exercise

7. Type in Example Program 3 at a computer terminal. Run this program for fixed values of A, B and values of N like 2, 4, 8, 16. Recall that the additional instruction given as

```
175 PRINT B↑4-B↑3+B↑2-7*B-A↑4+A↑3
          -A↑2+7*A
```

will print the exact value of the integral

$$\int_A^B (4x^3 - 3x^2 + 2x - 7)dx.$$

APPENDIX C ASSIGNMENT #1

For each of the following five problems, you will
(a.) set up the appropriate definite integral to solve the problem;
(b.) write programs in Basic that will approximate the definite integral by the Trapezoidal Rule and Simpson's Rule; and
(c.) use the programs on the Prime 300 and obtain approximate values.

Since you have yet to study enough integration techniques, you will not be able to evaluate the definite integral by the Fundamental Theorem of Calculus. Upon completion of Chapter 9 of the text, you will be able to do these same problems by the Fundamental Theorem of Calculus; then you will be able to compare the exact value of the definite integral with the approximate values you obtain in (c.).

#1.) The velocity of a car v(t), miles per hour, is inversely proportional to the square of the time t plus 1, where time t is measured in hours; i.e., $v(t) = k/(t^2 + 1)$. If the car passes a road side marker traveling at 45 miles per hour, find the distance it travels in the next 30 minutes.

(NOTE: This is one of the five problems of this assignment.)

APPENDIX D ASSIGNMENT #2

For each of the following four problems, you will

(a.) set up the appropriate definite integral to solve the problem;

(b.) write programs in Basic that will approximate the value of this definite integral by the Trapezoidal Rule and Simpson's Rule; and

(c.) use the programs on the Prime 300 and obtain approximate values.

Since you have not yet studied the derivatives and the antiderivatives of the elementary transcendental functions, you will not be able to evaluate the definite integral by the Fundamental Theorem of Calculus. Upon completion of Chapter 9 of the text, you will be able to do these problems by the Fundamental Theorem of Calculus; then you can compare the exact value of the definite integral with the approximate values you obtain in (c.).

#1.) The records of snowfall for many years in an area of upper New York State have the following average values:

0.41 inches for November 15
2.75 inches for December 15
4.22 inches for January 15
2.76 inches for February 15
0.38 inches for March 15

(a.) Show that the above data are reasonably well represented by the equation,

$$r = (4.2)(\sin(\pi t/150))^2$$

where r is the rate of snowfall in inches per day, t is measured in days starting at the beginning of November, and each month is assumed to have 30 days.

(b.) Use the above equation for r and write programs for the Trapezoidal Rule and the Simpson's Rule in Basic to estimate the total snowfall during the following periods:

(i) November 1 to March 30.
(ii) January 5 to January 27.
(iii) December 3 to January 27.
(iv) December 16 to March 2.

(NOTE: This is one of the four problems of this assignment.)

APPENDIX E ASSIGNMENT #3

Exercise #3 of UMAP Module #379 introduces a new function, ln, which we will soon be studying. This function is defined by a definite integral, where

$$\ln(x) = \int_1^x (1/t)dt .$$

In that exercise, you approximated ln(2) by applying the Trapezoidal Rule to the definite integral

$$\int_1^2 (1/t)dt .$$

Soon we will begin to study this function with regards to its continuity, derivative, some of its properties, and many of its applications.

This series of five exercises will introduce you to some of the properties of this new function. First, write a program in Basic for Simpson's Rule to approximate ln(x) for given values of x, $x > 0$.

#1.) For each of the separate sections below, use the given value of 300 to approximate the given value of the ln function. On a sheet of paper, record the results, each rounded off to two decimal places, for the given values of n, the number of subdivisions of the given interval of integration.

(a.) ln(2), ln(3), ln(6); n=1,000.
(b.) ln(2), ln(7), ln(14); n=1,000.
(c.) ln(5), ln(7), ln(35); n=1,000.
(d.) ln(5), ln(6), ln(30); n=1,000.
(e.) ln(3), ln(60), ln(180); n=1,000.
(f.) ln(4), ln(5), ln(20); n=1,000.
(g.) ln(2), ln(3), ln(5), ln(30); n=1,000.
(h.) ln(.2), ln(.3), ln(.06);n=10,000.
(i.) ln(.5), ln(.6), ln(.3); n=10,000.
(j.) ln(5/4), ln(4/5); n=10,000.

From the results of (a.) - (j.) above, make some conjectures for each of the following:

(i) $\ln(x \cdot y) = ?$
(ii) $\ln(x/y) = ?$
(iii) $\ln(1/x) = ?$

(NOTE: This is one of the five problems of this assignment.)

APPENDIX F ASSIGNMENT #4

Prior to studying calculus, you learned that π is approximately 3.14 or 22/7; these are two rational approximations for this irrational number.

#1.) You have also been exposed to π in the area of a circle formula, where the area $A = \pi r^2$ for a circle of radius r.

(a.) Explain why $\pi = 2\int_{-1}^{1} \sqrt{1 - x^2}\, dx$, using an area application of the definite integral.

(b.) Write programs in Basic that will approximate the value of the definite integral in (a.) above by the Trapezoidal Rule and Simpson's Rule.

(c.) From your results in (b.) above, determine various rational approximations of π.

(d.) When we study some of the techniques of determining antiderivatives in Chapter 10, we will see how to show that

$$\pi = 2\int_{-1}^{1} \sqrt{1 - x^2}\, dx .$$

(NOTE: This is one of the two problems of this assignment.)

STARTING A COMPUTER BASED LEARNING PROJECT

Barry MacKichan (presenter)
J. Mack Adams
Roger Hunter

Department of Mathematical Sciences
New Mexico State University
Las Cruces, N. M. 88003

Tel. (505) 646-3901

I. INTRODUCTION. A project funded by the Comprehensive Assistance to Undergraduate Science Education (CAUSE) program of the National Science Foundation began at New Mexico State University in July 1980. The project is an outgrowth of two earlier activities at the University: extensive use of self-paced instruction in lower division math courses, and use of microcomputers to teach a beginning course for computer science majors.

The specific objectives of the project are:

a) to develop primary computer-aided instruction (CAI) materials for a one semester course in trigonometry and a general education course in computer science;

b) to expand the adjunct CAI materials currently used in the first course for computer science majors; and

c) to develop adjunct CAI materials for a one semester course in discrete mathematics.

We are determined to build on the best existing work to accomplish these objectives. This paper describes the approaches we decided to use, our experiences with these, and the subsequent modifications of our methods. We hope this report will be helpful to groups who may be considering starting similar projects.

II. THE SELECTION OF THE DEVELOPMENTAL APPROACH. The first course for computer science majors has been taught for several years using Terak microcomputers and the UCSD Pascal system. Our success with this, as well as the trends in computer hardware, convinced us to stay with stand-alone microcomputers for our CAI materials. We have not, however, decided what particular brand of computers we will be using for delivery machines when large numbers of students will be using our materials. This uncertainty, as well as other considerations, has forced us to emphasize software portability.

Much of our approach is based on the experience of Alfred Bork's group at the Educational Technology Center at the University of California, Irvine. For example, we decided to implement our lessons in UCSD Pascal rather than in a CAI authoring language, and subsequent experience has vindicated this decision. Even before the project was actually funded, we had Alfred Bork present a workshop on authoring. Consequently, we purchased the authoring system of the Video Computer Learning Project at the University of Utah, led by Richard Brandt and Barbara Knapp.

After receiving funding, a literature survey of CAI development was conducted [Adm81]. This was done after our initial choice of approach, but it provided a measure of confidence about our decision.

III. INITIAL AUTHORING EXPERIENCE. We have now been involved with the authoring and implementing of lessons for almost a year, and feel that we have enough experience to comment on our approach, explain how and why we have modified it, and even give some advice to others who may be interested in starting out in CAI.

We started authoring lessons almost immediately after receiving funding. We were determined to begin immediately by relying on software implementation tools obtained from the University of California, Irvine and the University of Utah, avoiding a diversion of our efforts into software development.

a) Lesson specification: The basic approach of the project is that the authoring and implementation of lessons and their implementation as computer programs should be independent and distinct activities. Ideally, authors shouldn't feel constrained by implementation considerations. In fact, however, most teachers without computer experience underestimate what is possible, especially with graphics. Our project has benefited because there has usually been a programmer experienced in Pascal on graphics microcomputers on each team of authors.

We have followed the advice of Alfred Bork and others to work in small teams of authors and to specify the lessons with informal flow charts written on large sheets of newsprint. Our experience supports the importance of working in small groups (ours have usually been pairs of authors). The authoring teams provide interaction and ideas that generally improve a lesson. Philosophical discussions are frequent but they seem to benefit the quality of the lessons much more than impair the speed of writing. Also, meeting regularly with one or more authors gives the process a momentum which helps ease the more difficult times. Another advantage of authoring teams is that the less experienced members learn from others some of the capabilities of the computers which can be exploited, general approaches that have been successful in the past, and pitfalls.

The idea of using flowcharts to specify lessons has been useful, but we have found several shortcomings. Some minor improvements, such as using different colors for 1) the main flow, 2) the text which appears on the screen, and 3) comments, have helped. But more and more, as the lessons progess, we need a procedural type of "meta language." Frequently in the course of a lesson or help session, we need to recall a previously written dialogue or portion of a lesson. It is awkward to refer to a previous section of a flow chart with arrows or page numbers. We have been writing procedures which need to be recalled many times. This is particularly true for mathematics lessons, in which problems are solved by reducing them to several simpler problems, which in turn are solved by reducing them to several smaller steps, and so on. Our help sessions tend to recapitulate sections of previous lessons. Writing these as procedures not only simplifies the task of the programmers who are to implement the lessons, but also helps clarify the thinking of the authors.

Some of us feel the need for a lesson specification language that would be more flexible than flowcharting. Such a language is not intended to become a formal, machine implemented programming language, but rather an alternative to complicated flow charts with arrows jumping over ten pages. It must also be flexible enough that it doesn't limit the possibilities for a lesson. This language may evolve slowly as the authoring progresses.

b) Controllable worlds: One of the most effective ways we have found to exploit the special capabilities of the computer is to create "controllable worlds" or "machines" for the student. An example is a machine which graphs a function f(x) and moves the graph of af(bx+c) as the student changes the parameters a, b, and c. A bright and motivated student can learn the effects of changing these parameters (stretching and shifting the graph in the horizontal and vertical directions) simply by experimenting with the machine. A lesson using the machine is designed mainly to direct the student and force interaction so that the student learns inductively by doing. Another lesson in the trigonometry course uses an "angle machine" which the student uses to change angles and to acquire a quantitative feel for the size of angles.

Mathematics is certainly not an experimental science, but there is little doubt that mathematical insight comes from trying a large number of examples. Mathematically talented students may be able to visualize the graph of 3f(6x -2) from the graph of f(x), though most cannot. But almost everybody can benefit from watching the graph actually move as the parameters change. (The authors of the lesson caught some errors in the first version of the lesson after using the graphing machine themselves.) Controllable mathematical worlds implemented on graphics microcomputers create the possibility of bringing an empirical approach to mathematics.

The machine idea has also been used in a module on recursion for computer science courses. The student is given a "recursion machine" and is invited to change the recursion termination condition(s) and insert turtle graphics commands before and after recursive calls. By being able to vary these parameters at will, the student, in a very short time obtains experience in recursion that formerly took much longer. This approach extends that used by Ken Bowles [Bow77].

Lessons using machines or controllable worlds are entirely different from sections of a text book. Students are forced to interact, learn from experience, and verbalize what has been discovered rather than simply read. In fact, our approach using these machines evolved last summer when, for a period of several weeks, we had a commandment "Thou shall not press RETURN", in an effort to avoid the pitfall of creating lessons that consist of page after page of text on the screen, punctuated by the message, "Press RETURN to continue." We eventually decided that that message has a legitimate purpose, but living without it made us think much harder about the lessons and forced us to be more creative.

This experience in authoring has several implications for implementation. All the authoring languages that we have investigated are based on overly confined assumptions about what a CAI lesson is or should be. For example, it is impossible to say that student responses are either character strings, numbers, or positions on the screen. Our experience using controllable worlds is that a student response is the state of one of these controllable worlds after the student has manipulated it for awhile. There is no reason to believe that any data structure representable in a computer could not be a valid student response. Some of the modules we have already written have linked lists of pairs of coordinates as student responses, and it seems likely that directed graphs, trees, and even programs may be valid responses in future lessons.

If a module guides a student step by step through a problem, then each step may involve student responses which are simply numbers or phrases, but eventually, if students are to master the material, they will have to be able to solve a complete problem without any help from the computer. Ironically, when the student has less guidance from the computer, the necessity arises for allowing such a variety in student responses. The lesson module then has to avoid excessive control [Nie80] by analyzing the results to determine whether they are correct, partially correct, or completely wrong.

A basically linear approach, with branches allowed at questions based on the student response, is not sufficient for the lessons we have developed. In fact, many problems are best solved recursively by problem reduction. So perhaps the best way to implement a help session is to have the session call itself recursively.

Frequently the machines we use are logically at the center of the program. The student exits the machine and returns to it several times, which means the state of the machine must be preserved. Generally CAI authoring languages do not allow global variables sufficiently general to store these machine states. Nothing short of a powerful, general purpose programming language satisfies these requirements. We chose UCSD Pascal because of its availability and portability, our experience with it, and the good software available. We have not regretted this choice.

IV. INITIAL EXPERIENCE IN IMPLEMENTATION.

The cost of implementing lessons in a general purpose programming language is an increased programming effort. We have taken several steps to counteract this.

a) We have taken advantage of software developed at other universities, primarily the University of California, Irvine and the University of Utah. This software includes text and graphics editors which control the writing of text on the screen and the display of graphics and animation. We have been extremely fortunate in obtaining not only these software tools, but also valuable advice from both institutions.

b) We have developed locally a library of procedures which is shared by all our student programmers.

c) As the volume of this library of procedures grows, the problems of uniformity and assuring that all our programmers have the same current versions increase. A central hard disk storage unit connected to all development computers allows all the programmers to access the same library, assuring uniformity.

d) Several software tools to handle the routine programming chores are under development. A program has been developed that allows a programmer to lay out dialogue in ports on a screen interactively using a bit pad. The program produces Pascal statements and replays what the programmer has done by interpreting these statements. Extensions to the usual text editing program which allow large sections of text to be called from the library with a keystroke or a touch of the bit pad stylus are under development. These tools allow some of the speed of an authoring language for the routine portions of the programs.

The actual programming of the lessons is done by a group of talented students. During the starting phase of the project, estimates of programmer productivity were impossible to get, but we intend to collect this information during the coming year.

V. PROSPECTS. Some of the computer science and trigonometry lessons are being used this spring as supplementary material, and more extensive use is planned for the fall semester. Development of the discrete mathematics course has not yet started, but is expected to begin in the fall semester.

We are quite happy with the progress so far, and the university administration seems pleased as well and has given us support in addition to that indicated in the CAUSE grant proposal. This contribution and their support of workshops and software acquisition before the grant was awarded have been critically important in the progress achieved.

The initial promise of CAI in the 1960s was not matched by the results [Cha80, Nie80] , but we feel that this promise can now be realized by high quality lessons implemented on microcomputers. We feel that the obligation of those who work in the field is to produce the highest quality lessons possible, and to avoid cutting corners to increase productivity. A few well designed lessons will be of more value than any amount of low quality material.

REFERENCES

[Ada81] Adams, J. Mack and Clark, Barbara. A Selected Bibliography of CAI Material on Trigonometry, Discrete Mathematics, and Introductory Computer Science. NMSU CBL Project Report No. 1 (January 1981).

[Bow77] Bowles, Kenneth L. Microcomputer Problem Solving Using Pascal, Spinger-Verlag, New York (1977).

[Cha80] Chambers, Jack A. and Sprecher, Jerry W. Computer assisted instruction: Current trends and critical issues. Communications of the Association for Computing Machinery 23, 6 (June 1980).

[Nie80] Nievergelt, Jurg. A pragmatic introduction to courseware design. Computer 13, 9 (September 1980).

A COMPUTER-BASED DIALOGUE FOR DEVELOPING MATHEMATICAL REASONING OF YOUNG ADOLESCENTS

David Trowbridge
Alfred Bork

Educational Technology Center
University of California, Irvine
Irvine, CA 92717

INTRODUCTION

A project is being conducted by the Educational Technology Center at the University of California, Irvine to develop computer-based materials for junior high school students. The materials are in the form of modules dealing with topics in science and mathematics which run on personal computers. They emphasize skills of reasoning (i.e., formal reasoning in the Piagetian sense) such as reasoning with hypotheses, making observations and inferences, isolating and controlling variables, and reasoning with ratio and proportion.

Each module is devoted to a single topic and takes 1-2 hours to run in its entirety. The module consists of 5-10 separate activities each of which may be entered and exited independently.

The format of the learning modules is the Socratic dialogue. They have been written by teams of teachers who have considerable experience in working with students one-on-one. The modules provide a way of conducting an individualized tutorial with students who otherwise may not have the opportunity to work with a tutor due to limited staff or classroom time.

The topics of the modules include Area, Density, Functions, Graphing, Speed, Triangles, and Whirlybird. This paper describes in detail the module concerning the concept of area to illustrate the pedagogical approach we have taken.

EDUCATIONAL BASIS FOR THE PROJECT

A considerable body of knowledge generated by the work of Piaget and others suggests that the development of formal reasoning skills becomes possible around the age of 12-15. An equally substantive body of evidence suggests that formal reasoning skills are not developed automatically; in fact, they are strikingly absent in about 50% of college-age students.[1] Educators are generally in agreement that our educational system, all the way from middle school through college, is an appropriate place for teaching formal reasoning skills, and most would agree that the earlier the assistance is provided the better.

Many educational programs claim to be based on theories of intellectual development. We believe that present research in learning theory does not fully support any single approach to the development of materials to assist learning. However, two factors are favored by many different learning theories: (1) the need for an active learning environment, and (2) the need for individualized material allowing for greater diversity in student background.

The approach we have taken to meet these two needs in computer-assisted learning are based on the Mastery Learning Cycle.[3] Somewhat akin to Karplus's learning cycle, it has a few additional components.[2] Briefly, the Mastery Learning Cycle has elements of experience gathering or other motivational activity, instruction, application, and competency testing with feedback. The process is cyclical so that the results of competency testing can be used to route the learner back to one of the earlier elements for help if necessary.

DESCRIPTION OF ONE MODULE

As an illustration of how the Mastery Learning Cycle can be incorporated into computer-based learning materials, a module is described concerning the concept of area. This module consists of nine separate activities. The first three activities emphasize experience gathering; the next three provide instruction in certain key concepts; the next two activities involve applying these concepts; and the final activity is primarily a self-test for competancy. The activities in each of these groups are titled:

Experience Gathering:	1. Buying a Lot
	2. Measuring Area
	3. Practice in Measuring Area
Instruction:	4. Area with Two Different Unit Squares
	5. Standard Units of Area
	6. Recipe for Area
Application:	7. Area of a Parallelogram
	8. Area of a Triangle
Competency Testing:	9. Practice Finding Areas of Figures

The groups are not mutually exclusive. For instance, the activities of 4, 5 and 6 which have an emphasis on instruction, also contain elements of experience gathering, application, and testing. Indeed, a single group will often contain all the elements of the Mastery Learning Cycle. The following four sections describe the contents of each activity.

Experience Gathering— Activities 1, 2 and 3

The first activity, Buying a Lot, presents a familiar situation for thinking about area. A map is drawn of a residential subdivision, along with a square figure of variable size. The student is asked to adjust the size of the square until it appears to have the same area as a particular lot. This is an attempt to invoke whatever intuitive concept the student may already have about area in a friendly, informal way.

In the second activity, several aids necessary for measuring area by the method of counting squares are introduced. These are a grid of identical squares, an algorithm accounting for squares on the boundary of the figure (i.e., count them only if they lie more than half inside the boundary), and the notion of uncertainty in measurement. Help measuring the area of an irregular figure is provided along the way.

The third activity provides three examples of irregular figures: a pentagon, a lake, and a map of California. Abundant aid is available for those students having trouble applying the method of counting squares to any one of the examples.

This is a rather atypical approach to introducing the concept of area. Indeed, it is our experience that many students at the college level have never measured area by counting squares. Most have learned some rote procedure such as multiplying length times width to find the area of a rectangle, but when confronted with finding the area of an irregular figure they are at a complete loss. The module concerning area begins with irregular figures and the method of counting squares. Later, it introduces rules for shortcut methods of counting squares when the figure is a rectangle, parallelogram or triangle.

Instruction— Activities 4, 5 and 6

The next three activities treat some formal aspects of the concept of area: the consequences of changing the unit by which the area of a particular figure is measured, the relationship between standards of length and standards of area, and the operational definition of area.

In Activity 4, the area of a figure is first measured using a unit square specified by the program (Figure 1). Then students are asked to select their own unit square, either larger or smaller than the original unit square, and to make a prediction as to whether the number for the area will turn out to be larger or smaller than before. After making a prediction, the measurement is carried out to confirm or reject it.

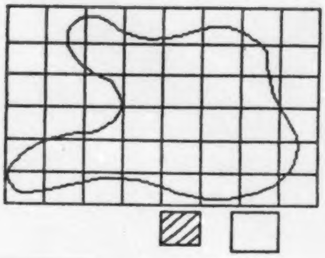


Figure 1: Measurement of area using arbitrary unit squares.

In the fifth activity, an attempt is made to draw from any familiarity students may have with terms for standard units of area. A rather sophisticated analysis of the students' responses is made to determine whether they have included, both "square" and "inch," or only "inch," or have given a common misspelling of an otherwise correct standard unit. The activity concludes with instruction about the number of square feet in a square yard using diagrams.

Activity 6, Recipe for Area, has a somewhat different format. It draws from all of the earlier material, attempting to review and summarize the steps that were required to obtain a numerical result for the area of irregular figures. Reflecting on one's thinking like this is a formal reasoning skill. The program suggests several plausible alternatives for the first step, second step, etc. in the process of measuring area and asks students to select the most appropriate sequence. The series of steps representing the operational definition of area are:

1) choose a unit square that is much smaller than the figure;
2) cover the figure with a grid of unit squart all squares entirely inside the figure;
3) count all squares entirely inside the figure;
4) estimate a number to account for the parts of squares along the boundary lying inside the figure; and
5) add together the two numbers you have obtained and call this the area.

Application— Activities 7 and 8

Activity 7 begins with a review of the rule for finding the area of a rectangle. A parallelogram is sketched, and by using a simple process that involves cutting off an "ear" of the parallelogram and transposing it to the opposite side, shows how the problem can be reduced to finding the area of an equivalent rectangle (Figure 2). Identifying what dimensions of the parallelogram may be called its base and height is somewhat subtle, so considerable help is provided at this point.

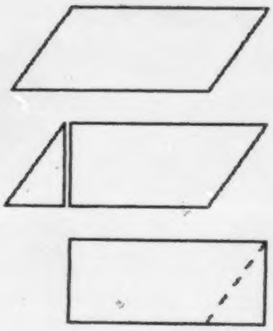


Figure 2: Construction of a rectangle from a parallelogram.

A second application of the concept of area is in Activity 8, Area of a Triangle. By construction it is shown that a triangle constitutes one half of a parallelogram. Again, the issue of just what constitutes base and height is treated not ex cathedra, but by discovery.

Competency Testing— Activity 9

The final activity provides students with an opportunity to test themselves on measuring the areas of several different figures: irregular figures, rectangles, parallelograms, and triangles. At each point of trouble, they are either given some immediate help (in case of initial or minor errors), or directed to an earlier activity in the module (in case of repeated or major errors).

STATUS OF THE PROJECT

Altogether, seven dialogues were written this past summer. First drafts, including all pedagogical specifications, were completed by writing groups which included at least one middle school teacher, an expert on intellectual development, and a staff member of the Educational Technology Center familiar with the technical details of computer-based materials.

Participation of junior high school teachers in each group was invaluable. They could easily identify vocabulary which would be too difficult for 7th and 8th graders and suggest alternative wording. Their experience was helpful in pointing out common conceptual and reasoning difficulties among their students, and they had a sense for the kinds of contexts students would find attractive or interesting. They had no difficulties interfacing with the authoring capabilities provided.

The flowcharts produced by the summer writing teams were reviewed in detail. Comments by reviewers have been very valuable in making revisions, rewriting sections of dialogue, and filling in gaps that were overlooked in the first draft.

Coding for four of the original seven modules (Area, Density, Speed, and Triangles) has been completed. The remaining three modules (Graphs, Functions, Whirlybird) are in the process of being revised and coded.

A group of a dozen or so student programmers write the programs in Pascal. Development is taking place on Terak 8510 micro-computers. With slight modifications in graphics routines, the programs will run on a variety of inexpensive personal computers.

After the student modules have been completely coded, we will begin development of corresponding teacher modules that will serve as a kind of computer-based teacher's guide for improving the effectiveness of the materials. The teacher modules will provide an outline of the overall structure of the module, a description of available help sequences, comments on the pedagogical purpose of each activity, and suggestions for incorporation of the computer materials into regular classtime activities.

The materials are aimed at developing formal reasoning skills that have wide applicability in science and mathematics instruction. We expect them to be useful for some high school and college students as well. We would expect college students who are weak in mathematics and elementary education majors who will teach these ideas in the schools to benefit from these materials. We would encourage others to use these learning modules and give us comments as to suggestions for improvements or indications of their effectiveness.

ACKNOWLEDGEMENTS

We would like to acknowledge the contributions of Arnold Arons, Barry Kurtz and John Pitre as authors of the Area module, and the efforts of those who have coded the various activities: Danielle Bernstein, Steve Bartlett, Elinor Coleman, Jeff Oswald, John Pitre, Naomi Salvador, Scott Tanner and Jim Zarbock.

REFERENCES

1. Arons, A. and Karplus, R. "Implications of accumulating data on levels of intellectual development," American Journal of Physics 44(4):396, 1976.

2. Karplus, R., et.al., SCIS Teacher's Handbook, Berkeley: University of California, 1974.

3. Kurtz, B. and Bork, A. "The mastery learning cycle in computer-based education," Proceedings of the Psychology of Mathematics Education conference, Grenoble, July 1981.

SWITCHING OVER TO THE MICROS

W. D. Maurer

George Washington University

The Department of Electrical Engineering and Computer Science of the George Washington University is switching its instruction in computer science from large and mid-size computers to microcomputers. It is anticipated that, starting in the fall of 1982, about 80% of our undergraduate course sections in software will use microcomputers exclusively. This paper is intended for those whose institutions are either in the midst of a change, or are at least contemplating it.

Various reasons can be offered for switching over to microcomputers. The most pervasive one is simple cost effectiveness. With the cheapest complete computer systems priced at about $500 each, the pressure to use such systems continue to mount, not only at educational institutions but throughout industry, government, and the military. A second reason is to prepare students for the computing world as they will know it, when microcomputers will be even more commonly encountered than they are today. A third reason is the advances in computer architecture. Those who are learning assembly language, for example, can use any of the well-known microcomputers to become acquainted with stack-oriented call and return statements and with status flags, and (on the 6502) with indirect addressing and relative addressing; none of these concepts are present on the IBM 370, 3031, or 4341. Likewise, microcomputer users can experiment with input/output at the machine level, something that is practically impossible for student users of a mainframe.

The disadvantages of the micros in comparison with the mainframes do not apply to student users. Limiting programs to 65,536 bytes of main memory, which can be a serious problem for graduate students, matters very little in elementary instruction. Only two years ago operating systems for microcomputers were so rudimentary that elementary students had serious problems with them; but today, most of the commonly used microcomputers have operating systems which allow assembly, compilation, and linkage editing. Flimsiness of certain microcomputer systems, such as the Radio Shack Model I, has been overcome by the introduction of heavier-duty models such as the Radio Shack Model III.

It will be assumed in the remainder of this paper that those who are in charge of teaching the computer science courses are not so self-indulgent as to be philosophically opposed to the basic idea of such a switchover. Re-education of the computer science faculty, of course, is required, particularly if they have been teaching Fortran (or some particular mainframe assembly language) exclusively. The switchover should not be done too fast -- it should be done a few courses at a time -- but it should not be done too slowly, either.

The only serious argument, in fact, that can be advanced against such a switchover, is: What should a student learn about mainframes? There are certainly some properties of mainframes which are not, currently, found on the micros (although that situation may change also, of course). There is floating-point arithmetic, hardware multiply and divide, 32-bit integer operations, string-handling in individual instructions, the cylinder concept, etc. We anticipate that, at least for a while, 20% of the instruction that students of software receive will continue to have to do with the properties of mainframe computers.

It will also be assumed that the proper number of microcomputers has already been purchased. It is easy to justify this kind of purchase for the elementary programming classes alone, since they usually have at least twice as many students as any of the other classes, and since it is here that the limitations of microcomputers matter the least. Once the purchases have been made, however, and once the students and faculty have had time to adjust to the operating systems of the purchased computers, the switchover can begin.

Flexibility is a key concept in any switchover of this kind. It is literally impossible to plan thoroughly as few as two years in advance, let alone five. The best philosophy is probably a very informal one: microcomputers are certainly becoming more important, therefore they should be used more and more in the classroom. The specifics -- what classes should be switched over the first year, the second, and so on -- should be planned year-by-year,with a computer committee meeting several times a year and making major decisions once a year. If there are procedural reasons not to do so, then a five-year plan may be constructed, but only with a firm understanding on the part of all those with real decision-making authority that major yearly revisions are to be expected, depending on new products introduced in the marketplace.

Textbook selection and purchase becomes very different when microcomputer software is taught. Many of the best textbooks in this area are published by tiny publishing houses that do nothing else, or almost nothing else; some are even self-published. Among these, in particular, are the works of Osborne, who produces the best reference works on microcomputers in general. But all of the necessary books will be available through computer stores. Washington has over 20 computer stores in its metropolitan area (not including Radio Shack outlets), so this task was easy. However, even for a school in an isolated part of the country, it is still wise to seek out the nearest large metropolitan area that has a computer store.

Except for Osborne's *An Introduction To Microcomputers*,expect to change texts for at least two or three courses each year. New and better texts are continually coming out; even Osborne has yearly editions that include new information,so that last year's books may not suffice for some of this year's courses.

Let us pass now to some of the specific courses that are commonly taught on computer subjects and see how these are affected by switching over to micros.

ELEMENTARY

If your school has an elementary course (some have elementary, PL/I, or the like, instead) then it is likely to be severely affected. For a long time Basic was the only well-known general purpose language offered on microcomputers. (So if you want, you can still teach Fortran on the microcomputers.) However, Basic is much more cost-effective. Most Fortran systems for microcomputers have been designed as full Fortran compilers, whereas most Basic systems for microcomputers have been designed as interpreters. Strangely, although we normally think of compilers as being more sophisticated than interpreters, the economics of microcomputers favor the interpreter. In any trade-off between space and time on a microcomputer, space is very precious (because of the small amount of main memory available) whereas time is so cheap as to be effectively free. Thus, for all but the longest-running programs (a chess program, for example), interpretation beats compilation, because the interpreter fits into a far smaller area of main memory.

So long as the difference between Fortran and Basic does not matter for one's elementary course, Basic is preferred. Fortran persists in one area -- an engineering school (such as ours) for those engineering students who are *not* majoring in computer science. Such students will enter a world in which the long-running program for carrying out engineering calculations and simulations is still the norm, and many of these programs are still written in Fortran (and very few in Basic). As of this writing, these students are still awaiting the switchover; that is, they are still using a mainframe.

ELEMENTARY

We do not have an elementary Pascal course, but this conversion should cause no trouble, since microcomputer Pascal is just as good as mainframe Pascal.

ASSEMBLY LANGUAGE

At one time, microcomputers had a reputation for lacking decent assemblers. Today, of course, mainframe assemblers are still much more powerful than microcomputer assemblers. But for teaching assembly language in microcomputers, the assemblers we have now are adequate.

Assembly language is far more important to the microcomputer programmer than to the mainframe programmer. On a mainframe, assembly language is now being discouraged and has been discouraged for quite a while now. On a microcomputer, however, it cannot be so easily discouraged, because of space limitations in most microcomputers.

Those who have been using IBM mainframes, as we have, may be especially dismayed by the prospect of going over to a microcomputer for assembly language. There are so many assembly language concepts which do not have counterparts in microcomputer assembly language that the courses are severely affected. Among these are multi-register operations, 32-bit operations, hardware multiply and divide, floating point instructions, and decimal instructions. At the moment, for this reason, our assembly language course is not entirely switched over. Our ultimate goal, however, is to move mainframe assembly language material into a second course in assembly language.

Assembly language programmers fall into two categories. One group is using it for purposes which were formerly implemented in hardware. Programming for logic design is distinctly different from programming for programming's sake; the programmer logic design must first know what logic design is about, whereas the pure programmer does not need to know such concepts.

Our solution to this problem has been to introduce a new course, a third hardware course requiring the one-year logic design sequence. Students in this course learn to implement hardware logic via microcomputer programming. At this time, this course has just been started.

HIGHER-LEVEL LANGUAGES

Any serious computer science student must become aware of the tremendous variety of higher-level languages such as PL/I, Pascal, Snobol, C, APL, and Forth. We teach the first three of these in one course. It is possible to teach C, and Forth on a microcomputer. (APL would probably require extra terminals, which is fine if you have the money for them; PL/M is arguably a reasonable subset of PL/I. Snobol seems to have been neglected, as far as microcomputers are concerned.)

It is arguable, however, that this is one area in which the student should really be learning about mainframes. The biggest advantage that mainframes will continue to have over microcomputers is their larger main memory size, and sophisticated programming language make good use of it. Of course, if a mainframe is not available, one must always remember that this does not preclude having a course on higher-level languages.

DATA STRUCTURES

All three approaches to studying data structures can be supported on microcomputers. First is studying how a data structure actually looks in machine language -- what pointers to move and how to move them, when you are inserting or deleting items of a list structure, or how to calculate and use hash table indices. Second is using a node-processing facility in a high-level language -- Pascal pointers and record types, PL/I based variables, pointers, and structures, or Snobol programmer-defined data types. Third and intermediate is using basic features of algebraic languages to construct nodes and pointers and manipulate them.

Pointer manipulation is an area in which mainframes do not have much of an advantage over even the 8-bit microcomputers. The HL register of the Z-80, the X register of the 6800, and the indirect addressing feature of the 6502 all facilitate handling pointers. Hash tables constitute an interesting area of study -- what is the fastest good hashing method on a microcomputer with no integer divide instruction? (One possibility for eight-byte identifiers: Form one byte of the hash code by taking the exclusive OR of all the bytes in the identifier. Form another byte of the hash code, if you need to, by taking the same exclusive OR, but rotating the partial result one bit to the left each time, just before the exclusive-OR operation.) Array manipulation on the 6502 is interesting because there are "short arrays" (of size 256 of less) and "long arrays" (of size greater than 256). Indexing in a long array takes much longer than it does in a short array, since the index registers (X and Y) are only eight bits long. In general, using a microcomputer for data structure courses provide many opportunities to learn the art of making do, which is necessary for any programmer, anyway.

COMPUTER GRAPHICS

Here the emphasis seems to be shifting from the theoretical to the practical. For a long time, experts in computer graphics developed highly sophisticated methods which were used only in special situations where more than the usual amount of funding was available -- research laboratories, television production, automobile design, and the like. As a result, computer graphics courses, although they proliferated, were almost never required. Microcomputer systems, however, have brought graphics within everyone's range. There is no longer any financial reason to do without graphics even in run-of-the-mill applications, and there are good arguments for making microcomputer-oriented graphics courses required in a computer science program.

Even in an environment where computer graphics was never studied, as such, up to now -- a community college, for example -- the graphic capabilities of modern microcomputers should be studied. In this area, however, the microcomputer textbooks are not sufficient. No microcomputer textbook that we have seen, no matter how much it says about the graphic capabilities of any specific computer, contains any material at all about two-dimensional transformations and matrix methods in graphics, let alone three-dimensional transformations and hidden-line algorithms. Reference to one of the standard works in the fields, such as Newman and Sproull.

OPERATING SYSTEMS

The time is long past when operating systems for microcomputers were rudimentary or nonexistent. Apple DOS, CP/M, and the like are complete operating systems having most of the features that any user would require. At the same time, microcomputer operating system problems are quite interesting because of the profusion of features that microcomputer operating systems do not, but could very easily, have.

It might be thought that the same argument would apply here as with compilers -- that is, that to have a decent operating system one really has to have a mainframe. This, however, turns out not to be the case. Programming language design, after all, is bounded only by the designer's imagination. A language such as PL/I, in which the idea is to have everything in one language, can quickly spawn a professor far too large even for minicomputers. The purpose of an operating system, however, is to operate a computer. It is easier--and therefore requires fewer and less complex programs, and a smaller amount of main memory--to operate a microcomputer than it is to operate a mainframe. Moreover, allowing students to try their hand at operating a mainframe can cause administrative difficulties, which would not apply in the case of a microcomputer. Converting the operating systems courses to microcomputers is greatly aided because microcomputer operating system manuals are typically far easier to understand than the corresponding manuals for mainframe operating systems.

NUMERICAL ANALYSIS

The biggest mistake one can make in setting up a numerical analysis course today is to assume that since microcomputers typically have no floating point instructions, they are therefore irrelevant to the numerical analyst. In the first place, digital flight control, an important new topic involves numerical calculations carried out on computers which quite often have no floating point instructions. But even the classical number-crunching problems become interesting exercises for microcomputers, as long as there is enough disk memory to handle intermediate calculations. For one thing, you can let a microcomputer run all night on a big problem, without spending any money.

This is not to say that mainframes should never be used in teaching numerical analysis. The numerical analysis course should probably involve both mainframe and microcomputer programming. One important use of microcomputers here is fixed-point calculations; any numerical analyst should learn how to do these (that is, keeping track of the binary point during multiplications and divisions, shifting registers to provide the optimum scale factor in every situation, and the like).

COMMERCIAL PROGRAMMING

We have commercial programming courses, but they are given in a separate department, which has its own curriculum committee. Nevertheless, the task of switching over a school to microcomputers may very easily involve a course in commercial programming.

Because Cobol is not found on any but the largest microcomputer systems, it is easy to jump to the conclusion that this is one course that should not be switched over. Again, this is being short-sighted.

First of all, commercial programming, more than any other kind of programming, is dominated by cost considerations. Even without Cobol, the commercial programmer--and, to an even greater extent, the designer of commercial programming packages -- will be greatly influenced by the low cost of microcomputers, and will use whatever is available. Basic is a common language for this purpose; it is even used sometimes on mainframes for commercial work. Assembly language has also been used, for much the same reasons as it is still used on mainframes.

It is true, of course, that historically Cobol has been considered easier to learn than Basic. Many people do not like Basic's algebraic notation and prefer the English-like statements of Cobol. Two factors, however, are changing this predilection. First, many more people than before are studying mathematics in high school and are familiarizing themselves with algebraic notation. Second, microcomputer system manufacturers are making a major effort to provide readable Basic manuals.

Again, do not suppose that commercial programming courses should stay away from mainframes entirely. There are certainly commercial programming problems which are simply too massive for microcomputers.

TYPING

Let us not forget this simplist of efforts -- even at the university level! Not that an engineering school should start up courses in typing, naturally; but one of the various "typing tutor" programs should be kept around for students to use when there is no one else on the computer. The productivity of a programmer, after all, is strongly affected by his or her typing speed.

For a school at which there are already courses in typing, a good case can be made for switching these over to microcomputers as well. The microcomputer can provide instruction that is simply impossible with a typewriter alone. Microcomputers are more expensive that typewriters, of course, and it is quite possible that the microcomputer will end up being used as an auxiliary tool in a typing course, rather than as the main tool. But the typing teachers should be made aware that computers and software systems are available (a typing teacher might have no other reason to visit a computer store).

RESEARCH

We come, finally to this most crucial of university activities. To what extent should research be done on microcomputers? I have a good friend who teaches computer science at a large urban university in the northeast; his Ph.D. is in mathematics. "I refuse to work with microcomputers," he tells me, "because it encourages students to have micro-thoughts." How pervasive is this kind of thinking, I wonder?

Research workers, of course, can use any computer they want to use and get access to. And some problems still require more memory or more software tools than a microcomputer can provide. But there are basic philosophical issues involved. If a computer program written in support of a research effort can be run on a microcomputer, it should be run on a microcomputer. Otherwise money will be needlessly wasted, and access to the use of the research will be needlessly restricted. Once this basic point is understood, researchers will automatically find themselves doing more of their work on microcomputers. And perhaps this, more than anything else, hastens the inevitable switching of a school's curriculum to the micros.

FACILITATING SMALL-SYSTEM MODULAR INSTRUCTION WITH AN INFORMATION SYSTEM

Amanda Evans
CACHE Corporation, and
Dept. of Chemical Engineering
The Univ. of Texas at Austin

David M. Himmelblau
Dept. of Chemical Engineering
The Univ. of Texas at Austin
Austin, TX 78712

ABSTRACT

The CHEMI modular instruction system combines features of information systems, stand-alone small computers, and computer-managed instruction with the objective of comprehensive coverage of the chemical engineering curriculum. Access to instructional and reference materials is provided through study recommendation procedures, keyword searching, and direct ID referencing. Interactive mastery testing capabilities are also provided by the CHEMI system.

INTRODUCTION

Throughout secondary and higher education we are observing an increasing use of small computers. Computer-based instruction provides reference and interactive learning material that can serve as the sole instructional basis of a course or, more likely, as a supplement to existing learning materials. Problem solving, drill, computation, simulation, and computer-assisted instruction (CAI) are all accomplished through student-computer interaction. With the rapidly advancing technology of microprocessors, instructional systems can now be designed to preserve the advantages of small computer systems as well as introduce some of the comprehensiveness and organization of larger systems. This paper describes one such computer application for college-level courses in engineering.

THE RELATION BETWEEN MODULAR INSTRUCTION AND INFORMATION SYSTEMS

Computer-assisted instruction is formed of modular materials in the same way a building is shaped by a structural framework. Although the computer serves as a tool that is easy to use and control, it demands a degree of rigor and organization that the average instructor is reluctant to face. Using computer-assisted instruction requires imagination and planning by the instructor as well as mastery of the subject material. This instructional system allows new ranges of materials to be covered and provides users and instructors more flexibility and free time.

One deficiency arising in most modular systems is the inability to rapidly move from one point to another within the set of materials. Another is the lack of rapid retrieval capabilities. A textbook or handbook has this advantage; that is, a reader can flip from one page to the index to another page or chapter without difficulty.

To be clearly effective, a modular instruction system must be supported by an information system that facilitates its use. This will induce students to select modular materials in preference to other sources of information. Increased accessibility can be provided by a system that retrieves information through indexing. Then, sequential and parallel relationships between modules can be retrieved on demand and used to select study paths as well as specific topics. Such a system might well be capable of keeping records and generating reports. Supplemental information (references to outside sources, definitions of terms, etc.) as well as instructional modules can be supplied to the user.

Here are a number of features that should be included in an information system for micro-and minicomputer-based modules in chemical engineering:

1) Help in choosing modules for study
2) Abstracts of outside sources
3) Computer programs for chemical engineering calculations
4) Glossary of terms and symbols
5) Interactive mastery testing
6) Record-keeping
7) Report generation

THE CHEMI SYSTEM

The CHEMI Project was initiated in July of 1975 via a grant by the National Science Foundation to the Computer Aids for Chemical Engineering Corporation (CACHE). The object was to produce three hundred single-topic, independent instructional modules spanning the seven key subject areas in the undergraduate chemical engineering curriculum. Each module is approximately twenty pages and covers a subject content equivalent to one hour of lecture. A second phase of the CHEMI Project has been funded to make the chemical engineering modules available on microcomputers to students and practicing engineers. In addition, five hundred abstracts referring to external sources of information not covered by the modules, and about one hundred computer programs for chemical engineering calculations have been prepared.

The CHEMI modules were originally written for off-line study. They were intended to be used either as remedial or self-paced instructional materials, or as replacements for sections of text that the instructor deemed unsatisfactory.

For the development and implementation of the CHEMI instructional system, the following configuration is in use:

1) PDP-11/23 with 256K bytes memory
2) UNIX operating system, version 7
3) Disk drive with 2 5MB double density platters
4) Tektronix 4025 graphics terminal
5) Dot matrix printer with graphics
6) Two alphanumeric terminals

The CHEMI system is designed to reside on small, stand-alone computers to allow local control over access and response times. Along with interactive mastery testing for each CHEMI module, the system software includes diagnostic and reference capabilities. The system can recommend sequences of modules for study, allow keyword searching for modules and abstracts of outside information sources, provide definitions of terms, teach users how to use the system effectively, provide computer programs for chemical engineering calculations, keep records of student progress, and generate reports on system use.(Fig.)

DIAGNOSTICS

Access to modules in the CHEMI system is provided by the diagnostic feature. Sequences of modules for users to study are recommended on request, based on subject preferences. This is done by presenting a user with topics associated with module sequences. If the topics are too broad, he may view sub-topics until he finds one with the desired scope over the subject area of interest.

Once a topic has been chosen, a user may examine the associated sequence or sequences of modules, and by adding or deleting modules can build a tailor-made study plan. To assist in this process, module summaries, objectives, and prerequisites may be viewed. Also, users may work pretests to see if they have the prerequisite skills needed for gaining maximum knowledge from certain modules.

When a suitable sequence has been worked out, the user may store it for future reference. The user's progress through the sequence can be recorded, and further modifications may be made, if desired.

REFERENCE

A second characteristic of the CHEMI information system is its reference capability. Much of the information in the CHEMI system can be accessed through keyword searching. The searches are based on stored indexing information, which includes term relationships such as "broader than," "narrower than," and "related to." The user may execute keyword searches to retrieve CHEMI modules, sections of modules, abstracts of outside sources, and chemical engineering computer programs.

In using a keyword search to retrieve information the user will most likely be referred to specific sections of the modules containing the desired information. He may examine these sections and have them printed out for later use. If the user employs a keyword search to find reference sources including non-instructional materials, he may retrieve abstracts of sources outside the system, as well as relevant module sections. The CHEMI system includes abstracts of articles from encyclopedias, handbooks, and journals. The articles were chosen to fill gaps in the existing contents and to provide greater detail for some topics.

The CHEMI computer programs are retrievable by keyword search also. The programs are short routines which perform typical chemical engineering calculations needed for small-sized problems. Currently, only code listings and documentation may be retrieved, but an attempt is being made to standardize the codes so that they may be executed as well.

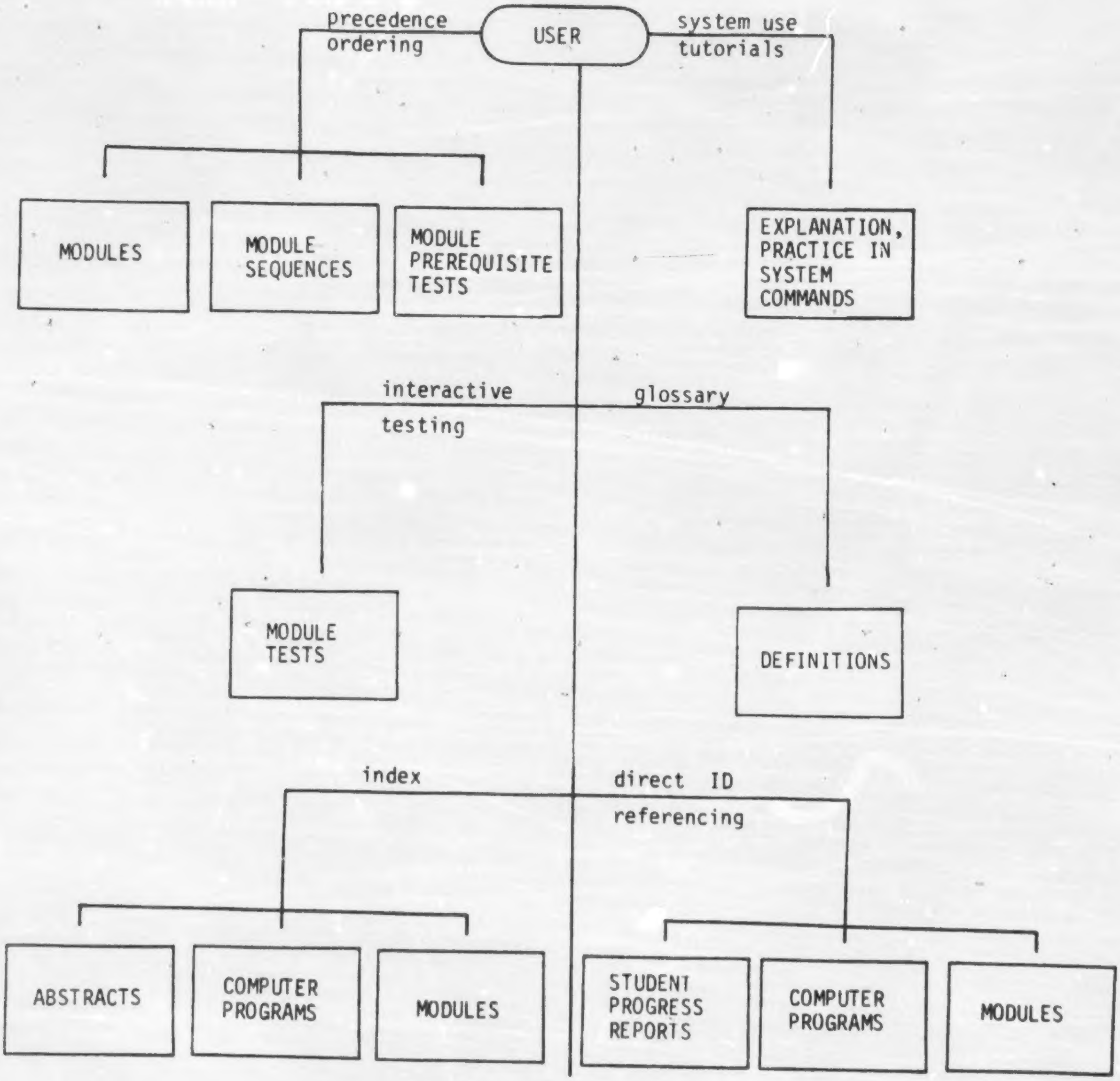

USER
precedence ordering
system use tutorials
MODULES
MODULE SEQUENCES
MODULE PREREQUISITE TESTS
EXPLANATION, PRACTICE IN SYSTEM COMMANDS
interactive testing
glossary
MODULE TESTS
DEFINITIONS
index
direct ID referencing
ABSTRACTS
COMPUTER PROGRAMS
MODULES
STUDENT PROGRESS REPORTS
COMPUTER PROGRAMS
MODULES

Figure: CHEMI Retrieval Methods

MASTERY TESTING

Not only does the CHEMI system provide greater access through diagnostics and keyword searching, but makes possible on-line study and interactive mastery testing as well. The CHEMI system includes an interactive mastery test for each module. The system can generate random numbers within problems and compute solutions to be checked against the user's answers. Test questions can also be chosen randomly from a pool of alternatives for a particular module. In this way, students may retake an unsuccessful test without seeing identical questions again.

The test questions conform to the objectives of the modules and correspond to a mastery level of comprehension. The CHEMI modules teach basic concepts; students are expected to demonstrate only the skills covered in each module in order to pass. The CHEMI system records the performances of students and generates reports for instructors.

SUMMARY

The CHEMI Project is designed to bring some of the advantages of large information retrieval systems to modular instruction on small computers. Retrieval via diagnostics and keyword searching can greatly increase the effectiveness of modular systems. Attention is being focused on tying together the set of CHEMI modules to form an acceptable instructional system that will improve the quality of chemical engineering education in universities and continuing education programs. The CHEMI Project is funded by National Science Foundation grant no. SED-7913021.

REFERENCES

1. Bailey, Daniel E. "Ingredients for Excellence in Computer-Based Education Systems." In: Harris, Diana. Proceedings of NECC 1979. NECC, 1979.

2. Chambers, Jack A. and Jerry W. Sprecher. "Computer Assisted Instruction: Current Trends and Critical Issues." Communications of the ACM. v. 23, n.6, June 1980, pp. 332-342.

3. Sugarman, Robert, "A Second Chance for Computer-Aided Instruction." IEEE Spectrum. Aug. 1978, pp. 29-37.

4. Van der Mast, Charles. "The Presentation of Courseware by Microcomputers Using a Modular CAI System." ACM Sigcue Bulletin. v. 13, n. 4, Oct. 1979, pp. 2-11.

PERSONAL COMPUTER SIMULATION PROGRAMS AS TEACHING AIDS:
A Successful Experiment
In Undergraduate Electrical Engineering Education

Aart J. de Geus, Leo R. Pucacco and Ronald A. Rohrer
Department of Electrical Engineering
School of Engineering and Applied Science
Southern Methodist University
Dallas, Texas 75275

Abstract - With realistic reservations about what can and cannot be done, personal computer-based software support is being developed in real time to supplement the introductory electrical engineering circuits course at the first semester sophomore level at Southern Methodist University. Pragmatic trade offs including cost considerations dictated what tasks can be done by personal computer and what must be done by student assistants so that software development could keep up with the lectures. Experience has indicated the approach to be both educationally and cost effective, as well as extendible to other courses.

I. Introduction

To address some critical issues regarding undergraduate electrical engineering education at Southern Methodist University, we use personal computer-based simulation programs in conjunction with the introductory sophomore-level electronic circuit analysis course. Interesting aspects of our approach have emerged because in much the same manner that our students address their homework problems we chose to take this path at the eleventh hour. Guided by a graduate student instructor a small group of senior undergraduate student assistants write weekly a set of simulation programs on personal computers. These simulation programs repeat, highlight, and extend the most important concepts covered in lectures and treated in the homework problems.

Lecture examples and homework problem solutions are programmed in Basic. By rendering all formerly fixed problem parameters variable by student users, these simple programs extend student instruction extraordinarily. On the average senior students spend about 25 hours per week developing these programs, including graphic, animation, and sound effects, the cost of which could later be amortised over the subsequent development of many similar programs.

The personal computer set-up is shown in Figure 1. Programs for the current and all previous weeks are available on (audio) cassette tapes. Usually a student starts a personal computer session by putting in appropriate values to check his or her homework answers. If the homework has been done correctly, this procedure is very fast. If there were errors, the computed answers become an effective guide to the sources of difficulty. Once the homework has been checked, a student can explore the circuits in several ways, such as using parameters that seem special, difficult, critical, or even nonsense. The

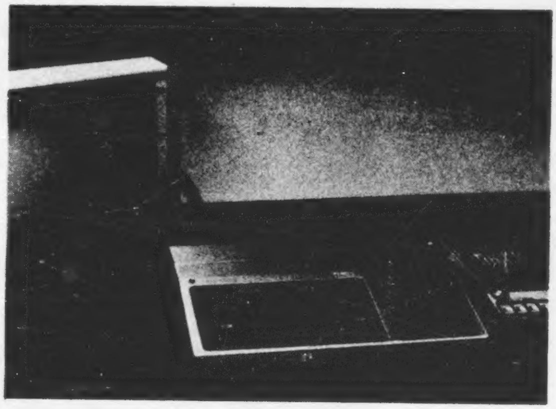


Fig. 1 The computer set-up consisting of a microcomputer, a TV monitor, and an audio cassette recorder.

personal computer can solve the circuit easily, so students do more problems since the correct answers are within easy grasp. Once a student has discovered that doing more rather than less is painless and profitable, then we think that we have a good engineer in the making.

II. Educational Impact

The following three aspects of an undergraduate engineering education are generally considered essential: theoretical understanding; appreciation of applications; and practical experience. Theoretical understanding and appreciation of applications ostensibly are provided by lectures and reinforced in homework problems; laboratories are often as close as undergraduate engineering students come to practical experience. That these classical approaches do not always work, or seldom work as well as one would like, is obvious to any engineering educator.

Common problems associated with classical undergraduate engineering courses include: large classes, particularly at critical introductory levels, so that stimulating question and answer sessions are difficult to provoke and sustain; different student backgrounds and motivations can render a course too difficult and time consuming for some students while simultaneously trivial and boring to others; and the amount of time available for problem solving and laboratory experience is almost insignificant compared to what is required in order to master the theoretical material of the lectures. The personal computer-based simulation programs appear to us to significantly alleviate these problems.

When a student checks the homework with the personal computer and finds mistakes, the values displayed help arrive at the correct answer in the correct manner. Hence, a vast majority of the trivial questions are circumvented, thus stimulating deeper and more meaningful questions. Moreover, senior student assistants who monitor the simulation laboratory are in a perfect position to provide immediate help and to provoke further thought: "What if. . ." Finally, simulation setups in the graduate instructor's and the lecturer's offices allow both of them to answer student questions in a nontedious environment, quickly converging on key points.

Differences in experience and motivation are smoothed out. The simulation programs allow advanced students to go beyond the course content and explore as yet unexplained cases. Some of these students have provided program improvements or even entire simulation programs of their own. For students having difficulties with the course, the personal computer is an ideal tool because of its patience. Moreover, it allows students to work with a vast number of cases in a very short amount of time so as to be able to develop a feel for circuit behavior. Students may use the simulation programs as often and as long as they wish so that they master each circuit type before proceeding to the next one.

Typically, there is not enough time for students to do enough homework to obtain a parametric appreciation for circuit behavior. To solve a circuit with a specific set of parameter values in no way imparts an appreciation of how circuit performance may alter as those values are altered. Students may work on paper one of a class of problems in order to master the solution techniques involved. Then, by playing with the values of the various parameters accessible in the personal computer simulation programs, they can quickly get a feel for what may be critical in circuit design and performance.

Circuit schematics of problems covered in this introductory course are shown in Figures 2,3, and 4. The sophistication of these

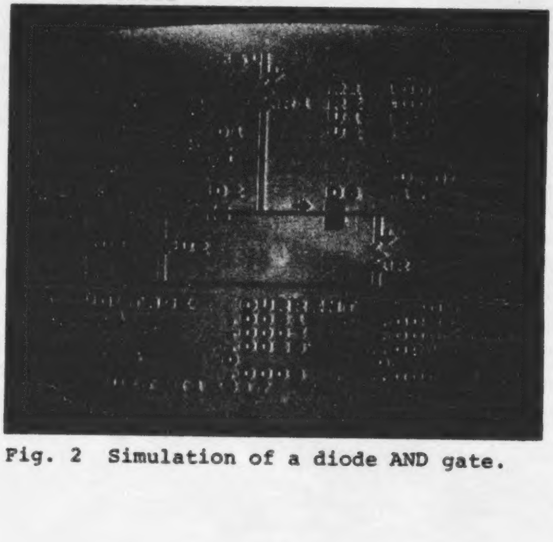


Fig. 2 Simulation of a diode AND gate.

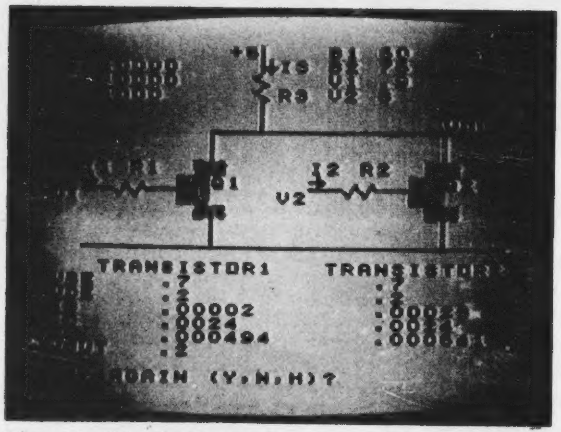


Fig. 3 Simulation of a transistor NOR gate.

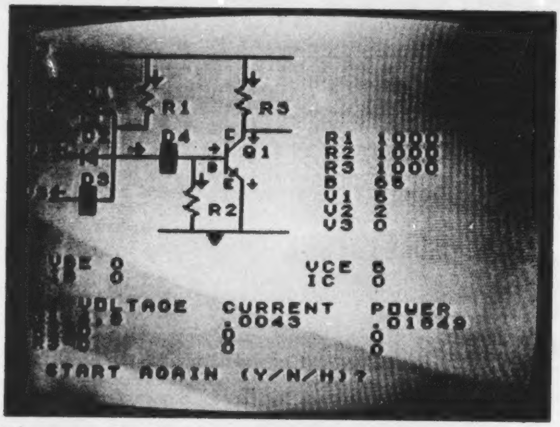


Fig. 4 Simulation of a DTL NAND gate.

circuits goes well beyond what students typically encounter in introductory electrical engineering courses.

III. Educational Effectiveness

There is qualitative and quantitative evidence that the use of personal computer-based simulation programs is an educationally effective adjunct to typical engineering teaching. Everyone--students, lecturers, senior student assistants, and graduate instructors -- benefit from this educational experiment.

The students taking the course are the focal point of this experiment. They are also the final judges of its effectiveness, and as program users are the best sources of the feedback essential to future course improvements. To obtain a crude measure of the simulation programs' effectiveness, we have not made their use mandatory. If a student feels that he or she is benefitting, a return visit is in the offing: attendance is feedback.

Once a student has used a given program, he or she is asked to complete an evaluation form (see Appendix). The answers to specific questions and the overall program grade guide the graduate instructor in up-grading the materials for subsequent use. Most important, of course, are the student comments that provide new ideas, suggestions, gripes, etc., in short, all that is needed to make the evaluation transcend into an evolution.

The principle student benefits appear to be the following. The homework problems can be checked on a self-paced basis; instant gratification is available. Additional problems are offered to stimulate further thought without requiring an inordinate amount of time. Students can rework problems with their own choices of parameter values and quickly obtain a feel for "what happens if. . ." Group work is stimulated as student attention inevitably turns from the passive CRT monitor screens to the activities of each other. Senior student assistants clear up basic problems as well as probe more advanced ones. The simulation program set as a reference highlights in a unique and concise manner the key points of the course. Quick course review is easily achieved by running through the set of simulation programs

The lecturer determines the course contents independently of any considerations for its subsequent simulation using personal computers. Then in consultation with the graduate instructor, the lecturer decides which key points are best served by the personal computer simulation. The simultaneous software development requires a clear view of what is important as the lecturer rethinks his or her educational approach. (We have not yet had an opportunity to experiment with teaching a course with simulation software that already exists; that is to be the challenge of next semester.) The personal computer helps the students to answer the trivial questions for themselves, leaving for the sometimes harried instructor of a large class the essential business of dealing with the more involved questions that students may be moved to ask. When the simulation programs are available in advance, they can aid the lecturer in preparing homework and examination problems. A simulation set-up in his or her office aids substantially a lecturer attempting to answer individual student's questions.

The senior assistants first develop the simulation software in real time conjunction with the lectures and later assist the students in its use. To design an error-free program around a specific electrical circuit entails complete understanding of that circuit; this automatically provides the senior assistants with an excellent in-depth review of the basic electrical engineering topic in question. The real time aspects of the software development schedule have enforced excellent engineering discipline on the senior assistants. The necessity of developing the software on a small and inexpensive machine, constantly aware of its limitations, forces him or her to cope with practical engineering constraints. Program development involves a group effort, a requirement rarely encountered in a standard engineering curriculum. Therefore, both their communication and organization skills have been refined in a professional and pragmatic manner. The software is designed for a student audience that does not hesitate to provide direct and meaningful feedback on the effectiveness and ease of use of the programs. In all, the senior assistants are subjected to a real engineering experience in a reasonably controlled environment.

In our scheme the graduate instructor is the key course coordinator. In many ways he or she comes to accrue most of the benefits that fall also to both the lecurer and the undergraduate assistants. In addition, the graduate instructor bears the formidable management responsibility, dividing software development responsibilities among senior assistants beset with diverse course schedules and enticing extramural activities competing for their attention. Finally, he or she must judge student feedback and translate it into practical modifications and future enhancements.

The results of our one semester experiment have been both positive and encouraging; these conclusions being based on four sources of direct feedback. First, the weekly questionnaires (see Appendix) containing 738 replies from 112 individual students show that the effectiveness and value of the personal computer-based simulation programs as judged by the students achieved a score of 3.39 out of a possible 4.00. Second, the results of a questionnaire distributed at the end of the semester revealed that out of 116 students who returned it, 108 stated without reservations that the personal computer was a definite help, a good idea and, most important, should be continued and expanded. Third, a comparision of the course averages of all students showed that those who used the personal computer achieved a significantly higher average than those who didn't. Finally, as a direct result of their experiences with the personal computer and the programs made available, several students were inspired to develop their own software.

IV. Cost Effectiveness

So far this experiment has been applied to a single one semester course enrolling slightly more than 140 students. Typically, to handle this number of students would have entailed five or six separate sections costing one to one-and-a-half full years of equivalent lecturer salary at approximately $30,000 - $45,000. We have used one lecturer full time for one semester at an equivalent cost of approximately $15,000. This project cost an additional $6,200 which breaks down as follows: one half-time graduate instructor at $3,000; 300 hours of senior assistant programming at $5/hour for $1,500; 240 hours of senior assistant laboratory supervision at $5/hour for $1,200; supplies (mainly tapes and disks) for developing and duplicating programs for approximately $500.

Even were we to purchase the ten personal computers used at $500 apiece, we would still be well ahead financially. Moreover, much of the cost itemized above is to be amortised over subsequent offerings of the same course. We estimate that one lecturer can easily address two such courses with the help of one-quarter time graduate instructor and only 240 hours of senior assistance, more than cutting the costs in half. In this era when the ratio of interested undergraduate students to qualified instructors is growing at a prodigious pace, personal computer-based simulation programs may be the sole salvation of engineering education.

V. Further Developments

We feel that the entire undergraduate electrical engineering curriculum can be enhanced by including personal computer-based simulation programs. Following the success of our one semester experience, next semester we plan to repeat the first course while providing supplementary software for two more.

Very shortly we expect the prices of personal computers to fall to the point where they can be removed from the laboratory environment and be home in the hands of every student. As a result, the role of laboratories, textbooks, and even lectures will have to be reconsidered, and engineering education must embrace the fruits of a constantly advancing technology. It is the responsibility of engineering educators to make innovation their ally instead of their enemy.

VI. Acknowledgements

This experiment could not have been undertaken at Southern Methodist University without the generous assistance of Texas Instruments, Inc. TI provided both the prototype hardware and the generous technical assistance of John d'Angelo and Alfred Riccomi of its Central Research Laboratories. Deans F. Karl Willenbrock and William F. Leonard of the School of Engineering and Applied Science have been unwavering in their support of our expenditure of resources, time, and energy in the service of undergraduate education. Both the sophomore students and the senior assistants have been stimulating subjects who quite frequently have raised our educational sights considerably.

Appendix

QUESTIONNAIRE

NAME__________________________

TIMESLOT_______________________

CLASS SECTION__________________

This questionnaire is intended to evaluate the "processor-usage." Your suggestions, criticisms, remarks, experiences. . . are most important to us and we need your feedback to set up an interesting and enjoyable course. Thanks for your help!

PROGRAM # ____

Did you use the program? YES ___ No ___

EVALUATION Give scores A, B, C, D, or F.

Coordination with course: ___
Coordination with homework: ___
Coordination with 2151 Labs: ___
Did you learn something: ___
Did it help you with the theory: ___
Did it help you with the problems: ___
Availability of assistance: ___

Length:
too short ___ about right ___ too long ___
Difficulty:
too easy ___ about right ___ too difficult ___

OVERALL SCORE ____
Suggestions, remarks. . .

COLLEGE MATHEMATICS

Robert F. Maurer
Timothy Donovan
Wayne F. Mackey
Susan Lindsay
Harold Harp
Thomas Wallgren
LaRuth H. Morrow

ABSTRACT: Calculus Demonstration Programs

Robert F. Maurer and Timothy Donovan,
The Pennsylvania State University,
Mont Alto Campus, Mont Alto, PA 17237

The mathematics faculty in the Commonwealth Campus System of the Pennsylvania State University are cooperating in the developing computer graphics programs demonstrating various topics in calculus. The programs are all written in Applesoft Basic for the Apple II Plus minicomputer.

The variety of programs presently evolving includes:

1. Graphs of functions of a single variable;
2. Demonstration of the derivative;
3. Graphs of relations $(F(x,y) = 0)$;
4. Graphs of functions in polar coordinates;
5. Graphs of 3-dimensional surfaces; and
6. Numerical solutions of differential equations.

Additional topics for which programs are being written or are contemplated include the definition of limit, the definition of the definite integral, infinite series, and areas in polar coordinates.

ABSTRACT: Computer Use in Community College Mathematics Education

Wayne F. Mackey, Susan Lindsay, and Harold Harp, Johnson County Community College, College Boulevard at Quivira Road, Overland Park, KS 66210

Currently there are four types of computer use in community college mathematics education: tutorial program, computer-managed instruction, computer as automated teacher, and computer as visualization aid. In this presentation we will define and explain each type, discuss its benefits to teacher and student, and either discuss or demonstrate a sample program. In addition, we will present the results of two pilot programs at Johnson County Community College which study the effectiveness of computer use in mathematics instruction.

We will suggest ways in which these types of computer use might be combined and enhanced to build new applications for mathematics instruction. The programs we will discuss and demonstrate are written in Applesoft Basic to run on an Apple II microcomputer.

ABSTRACT: Two Pascal Programs for Beginning Calculus

Thomas Wallgren, Millikin University,
Decatur, IL 62522

The following programs will be discussed in this presentation. The first program illustrates the definition

of Riemann integral through various ways of forming Riemann sums (upper sums, lower sums, evaluation at midpoints, and five more methods). The second program draws Poincare phase-plane trajectories for autonomous pairs of differential equations.

ABSTRACT: Software Used in Developmental Mathematics

LaRuth H. Morrow, 1317 Navaho Trail,
Richardson, TX 75080

Richland College offers a series of courses using self-paced tutorials, developing mathematics skills in adults not ready to take college mathematics courses. Recently, software has been written at Richland to enhance these courses, particularly the lower levels. These programs are the first computer-based training aids at Richland in developmental mathematics. They include interactive training, reinforcement, and testing of skills. The programs have been written for the Texas Instruments 99/4 computer and use its graphic and color capabilities. A demonstration of the programs and an assessment of their suitability for use in developmental mathematics courses will be presented.

Author Index

Adams, J. Mack 320
Alley, Lebert R. 153
Anderson, Cheryl A. 292
Anderson, Ronald E. 240
Anger, F. D. 66
Arenson, Barron 114
Arons, Arnold 220
Autrey-Hunley, Carolyn 276

Bandelier, Nellie 269
Barnes, M. 311
Barth, Paul 206
Bartholomew, Rolland 228
Bass, George M. Jr. 37
Beltran, M. Beatriz 194
Benenson, Harold 245
Bertsche, Steven V. 113
Billings, Karen 279
Bonar, Jeff 206
Boody, Charles, G. 280
Bork, Alfred 78, 179, 220, 324
Brandt, Richard C. 78
Brooks, Connie J. 194
Breitler, Alan 34
Burke, Chauncey 89
Burns, Hugh 105

Callahan, Robert E. 89
Cambell, James 264
Carroll, Tim 302
Christensen, Margaret 35
Cleveland, Linore 154
Collea, Francis 220
Cook, Robert N. 198
Crossland, Cathy L. 99

Dalton, Diann 193
de Geus, Aart J. 336
Donovan, Timothy P. 281, 341
Durrett, John 284
Dworak, Paul E. 136

East, J. Philip 242
Eerkes, Gary L. 45
Ehrlich, Kate 206
Elliott, Roy 83
Evans, Amanda 332
Evans, Richard W. 254
Evans, Selby H. 259
Eveland, S. A. 38

Fauley, Franz E. 278
Franklin, Stephen 179, 220
Freund, Robert 284

Gale, Douglas S. 277
Garson, James W. 311
Gorman, Henry Jr. 81
Gottschalk, Doris Duncan 83

Haiduk, H. P. 192
Harp, Harold 341
Hasselbring, Ted S. 99
Hazan, Paul L. 244
Heller, Helen 38
Heller, Paul 276
Heller, Rachelle S. 74
Helwick, Katherine L. 193
Herman, Eugene A. 18, 150
Himmelblau, David M. 332
Hooper, Kristina 150
Horton, Robert L. 45
Hunter. Roger 320

Johnson, Nancy 302
Judd, Dorothy H. 297
Just, Steven 245

Katz, Martin 179
Keefer, Roland J. 127
Kehler, T. P. 311
Kolodny, Nancy H. 59
Kuhn, Wolfgang 142
Kurtz, Barry L. 220

LaFrance, Jacques 80
Lagowski, J. J. 184
Lewis, Coleta L. 81, 272
Lindsay, Susan 341
Lindstrom, Peter A. 313
Little, Elizabeth R. 276
Lola, Pat 81
Lorton, Paul. Jr. 142
Luce, Larry 49
Luce, Thomas G. 121
Luehrmann, Arthur 150

Mackey, Wayne F. 341
MacKichan, Barry 320
Maleske, Robert T. 12
Marks, Gary H. 228
Martin, C. Dianne 74
Mathis, Robert F. 310
Maurer, Robert F. 281, 341
Maurer, W. D. 327
McIsaac, Don 238
McNelly, John 179
Miles, David 312
Mims, Ted 280
Mitchell, Stephen 111
Moore, Freeman L. 287
Morrow, LaRuth H. 342
Motto, John W. 24
Mozes, George 152
Murphy, Kay 82

Neideffer, Jerry D. 259
Nichols, Raymond 109
Nickles, Herb 191

Ortega, Jacqueline 94
Ott, Gene 59
Overall, Theresa 81
Owles, V. Arthur 112

Parrish, James W. 113
Pleva, Michael A. 152, 175
Plummer, Robbie 190
Pogue, Richard E. 309
Polin, Glenn 243
Pollak, Richard A. 78
Powers, Michael J. 112

Prather, Ronald 39
Pucacco, Leo R. 336

Rabia, Ahcene 39
Radspinner, Diana B. 193
Ries, Roger R. 37
Rodriguez, R. V. 66
Rogers, Jean B. 242
Rohr, John A. 161, 195
Rohrer, Ronald A. 336
Rubin, Eric 206

Schenk, Robert 4
Scott, Janet M. 193
Settle, Frank A. Jr. 152, 175
Shankman, Brian 109
Silvia, John E. 1
Skelton, W. A. 115
Slaton, Deborah C. 193
Smith, Arlene M. 193
Smith, Vicki S. 194
Soloway, Elliot 206
Southwell, Michael G. 38
Spain, James D. 113, 225
Suits, Jerry P. 184
Sustik, Joan 78
Swigger, Kathleen M. 264

Taylor, Heimy F. 132
Taylor, Reed D. 169
Tinker, Robert F. 309
Trowbridge, David 324
Turner, A. J. 36
Turner, Ronald C. 127

Vazsonyi, Andrew 86

Wachtel, Edward E. 240
Walker, R. S. 115
Wallgren, Thomas 341
Ward, John A. 8
Watt, Daniel H. 82, 279
Weiler, Sally J. 152
Weissman, Jessica 109
Woolf, Beverly 206
Wood, Lewis J. 94
Wood, Susan M. 37
Woolverton, Michael W. 169
Wozniak, Matthew 114
Wright, Edward B. 311

Zinn, Karl 239
Zwiener, Catherine 284

Subject Index

This index lists by topic papers published in the Proceedings of NECC (1979) and NECC/2 (1980).

- Business/Economics
  - Computer-Assisted Instruction for Large Classes in Economics
    - Paden and Barr, 1979, p. 227
  - Computer Science and MIS College Students: An Investigation of Career-Related Characteristics
    - Jordan, 1980, p. 7
  - Force Feeding SPSS in Market Research and Analysis at Hampton Institute
    - Wehrle, 1980, p. 12
  - Short-Run Forecasting of the U. S. Economy
    - Bowman, 1980, p. 16
  - Simulating the Great Depression in Introductory Macroeconomics
    - Schenk, 1979, p. 179
  - A Small Instructional Macroeconomic Model of the United States Economy
    - Werner, 1979, p. 189

- CAI/CMI/Computer-Based Education
  - Computer-Based Education: Research, Theory, and Development
    - Hall, 1979, p. 7
  - Computer-Managed Education: Providing Quality Education on a Large Scale
    - McGlinn and Carmony, 1979, p. 44
  - Ingredients for Excellence in Computer-Based Education Systems
    - Bailey, 1979, p. 2
  - The Non-Technical Factors in the Development of CAI
    - Mocciola, 1980, p. 236
  - An On-Line Directory of CAI Resources
    - Swigger and Maner, 1979, p. 48
  - The State of the Art of Instructional Computing: A Report from CONDUIT
    - Johnson, 1979, p. 14
  - A Successful Computerized Management System for Individualized Instruction
    - Stoaks, 1979, p. 52
  - Toward the Development of a Computer-Managed Instructional System in Pharmacy, Based Upon Integrated Database System Concepts
    - Henkel and Williamson, 1979, p. 360

- Computer Games in Instruction
  - Shall We Teach Structured Programming to Children?
    - LaFrance, 1980, p. 261
  - Structured Gaming: Play and Work in High School Computer Science
    - Moshell, Amann, and Baird, 1980, p. 266
  - Tapping the Appeal of Games in Instruction
    - McKay, 1980, p. 271

- Computer Laboratories in Education
  - The Computer Lab of the 80s
    - Brown, 1980, p. 256
  - The Education Technology Center
    - Bork, Franklin, and Kurtz, 1980, p. 258
  - Microcomputers in the Teaching Lab
    - Tinker, 1980, p. 250

- Computer Literacy
  - A Byte of BASIC
    - Hopper, 1980, p. 62
  - A Case for Information Literacy
    - Schimming, 1980, p. 58
  - Computer Literacy for Liberal Art Students: The Case of Computer Mapping
    - Clark, 1979, p. 107
  - A Computer Workshop for Elementary and Secondary Teachers
    - Dersham and Whittle, 1980, p. 65
  - Microcomputers and Computer Literacy: A Case Study
    - Ellison, 1980, p. 68

- Computer Science Education
  - Development of a Resource Center for Introductory Computer Science
    - Lee, 1979, p. 135
  - Initial Evaluation Results for an Introductory Programming Course without Lectures
    - Hazen, Daly, Embley, Nagy, and Prange, 1979, p. 150
  - Required Freshman Computer Education in a Liberal Arts College
    - Wetmore, 1980, p. 139
  - Systematic Assessment of Programming Assignments
    - Bishop, 1980, p. 147

- Computer Science Topics (see also Structured Programming)
  - A Computer Software Technician Program at Portland Community College
    - Hata, 1980, p. 287
  - Computer Support for Courses in Data Entry and Word Processing
    - Sunkel, Rickman, and Hobbs, 1979, p. 205
  - Data Structures at the Associate Degree Level
    - Dempsey, 1980, p. 152
  - A Descriptive Approach to the Teaching of Computer Organization
    - Vranesic, 1979, p. 101
  - Development of Communication Skills in Software Engineering
    - Beidler and Meinke, 1980, p. 143
  - A Secondary Level Curriculum in System Dynamics
    - Roberts and Deal, 1980, p. 281
  - What Programming Language, If Any, for Beginners?
    - Hughes, 1979, p. 141

- Computing Curricula
  - A Computer Science Major in a Small Liberal Arts College
    - Mayer, 1980, p. 290
  - An Educational Program in Medical Computing for Clinicians and Health Scientists
    - Hybl and Reggia, 1980, p. 276
  - The Proper Role of the Computer Science Program at a Small Undergraduate Liberal Arts University
    - Horton, 1979, p. 197
  - Required Freshman Computer Education in a Liberal Arts College
    - Wetmore, 1980, p. 139
  - A Segmented Structure for Remedial Graduate Computer Science Instruction
    - Pozefsky and Tharp, 1979, p. 201

- Elementary School Computing Programs
  - A Comparison of the Problem-Solving Styles of Two Students Learning LOGO: Computer Language for Children
    - Watt, 1979, p. 255
  - Language in the LOGO Computer Culture
    - Solomon, 1979, p. 250
  - OZNAKI and BEYOND
    - Cohen, 1979, p. 170
  - The Scarsdale Project: Integrating Computing into the K-12 Curriculum
    - Sobol and Taylor, 1980, p. 155

- Engineering
  - Computer-Augmented Video Education in Electrical Engineering at the U. S. Naval Academy
    - Lim, Hagee, and Pollak, 1980, p. 90
  - Computers: New Avenues for Engineering Students at the Community College
    - Brillhart and Shawn, 1979, p. 21
  - On Introducing Computer-Aided Engineering Techniques into an Established Mechanical Design Curriculum
    - Sutherland and John, 1979, p. 18

Faculty Considerations
    Computer Training for the Faculty at Drew University
        Metzler, Nelson, and Carroll, 1979, p. 378
    Transitioning Instructor Roles in Computer-Based Environments: A Preliminary Program for Improving Instructor Attitudes
        Dobrovolny, McCombs, and Judd, 1979, p. 380

Graphics
    Computer Graphics as a Lecture Aid in Undergraduate Mathematics
        Porter, 1979, p. 237
    Tessellations and the Teletronix 4051
        Tanis, 1979, p. 233
    Using Graphics for CAI with a Minicomputer
        Ward and Rudnick, 1979, p. 271

Handicapped
    A Computer-Assisted Instruction System for the Blind and Visually Impaired
        Ballenger, 1979, p. 274
    A Microcomputer/Videodisc CAI System for the Moderately Mentally Retarded
        Thorkildsen, Bickel, and Williams, 1979, p. 285
    SPEDINFO: A Decision-Making Support Feature for the PLATO System
        Cronin and Rizza, 1979, p. 280

Health
    An Adverse Drug Interaction System: Its Design, Operation and Use
        Jaufmann, Williams, and Barletta, 1979, p. 354
    Design of a CAI Tool to Generatively Teach Etymology
        Garcia and Rude, 1979, p. 371
    Developing a Teaching/Learning Experience for Nurses in Fundamentals of Computer Programming Preliminary to Nursing Research
        Erat and McGrath, 1979, p. 316
    An Educational Program in Medical Computing for Clinicians and Health Scientists
        Hybl and Reggia, 1980, p. 276
    A Problem-Solving Model in Continuing Education for Dental Practitioners
        Lipson, Keith, and Jones, 1979, p. 301
    Toward the Development of a Computer-Managed Instructional System in Pharmacy, Based Upon Integrated Database System Concepts
        Henkel and Williamson, 1979, p. 360
    A Training Program in Health Computer Sciences
        Ellis and Gatewood, 1979, p. 311

Humanities
    Creativity through the Microcomputer
        Bass, 1980, p. 38
    From a Theory of Reading to Practice via the Computer
        Johnson and Baldwin, 1980, p. 46
    Generative CAI for Foreign Languages
        Decker and Rice, 1979, p. 326
    Giving Advice with a Computer
        Garson, 1980, p. 42
    Melodic Perception Development and Measurement through Computer-Assisted Instruction
        Ray and Killam, 1979, p. 329
    Non-Harmony: A Vital Element of Ear Training in Music CAI
        Groom-Thornton, and Corbet, 1980, p. 54
    The Stanford Logic Course: Design, Dissemination, and Demonstration
        Ager and McDonald, 1979, p. 335

Mathematics
- Computer Applications in a Finite Mathematics Course
  - Piegari, Abernethy, and Thorsen, 1980, p. 179
- A Computer-Assisted Course in Biomathematics
  - Wong, 1980, p. 184
- Computer Graphics as a Lecture Aid in Undergraduate Mathematics
  - Porter, 1979, p. 237
- Computer Symbolic Math
  - Stoutemeyer, 1980, p. 194
- An Examination of the Use of Computers to Increase Student Motivation and Understanding in Relation to Complex Mathematical Models
  - Dennis, Stevenson, and Colella, 1979, p. 194
- A Method for Experimenting with Calculus Using CAI
  - Anger and Rodriguez, 1980, p. 171
- Tessellations and the Teletronix 4051
  - Tanis, 1979, p. 233

Micros and Minis
- The AVID System: A Multi-User Minicomputer-Based System for Computer-Based Education
  - Scholz and Sloane, 1979, p. 264
- Computer-Based Instruction for the Public schools: A Suitable Task for Microprocessors?
  - Taylor, 1980, p. 223
- Considerations and Guidelines for Developing Basic Skills Curriculum for Use with Microcomputer Technology
  - Caldwell, 1980, p. 31
- Creativity through the Microcomputer
  - Bass, 1980, p. 38
- The ISU Microsupport Net: A Prototype System for Personal Computing in Education
  - Horton, 1979, p. 268
- Microcomputers and Computer Literacy: A Case Study
  - Ellison, 1980, p. 68
- Microcomputers as Laboratory Instruments: Two Applications in Neurobiology
  - Olivo, 1980, p. 81
- Microcomputer-Assisted Study and Testing System
  - Garraway, 1980, p. 200
- Microcomputer-Generated Videotapes
  - Cooke and Martin, 1979, p. 347
- Microcomputer Software Development: New Strategies for a New Technology
  - Kehrberg, 1979, p. 343
- Microcomputers in the Teaching Lab
  - Tinker, 1980, p. 250
- Using Graphics for CAI with a Minicomputer
  - Ward and Rudnick, 1979, p. 271

Minority Institutions
- Academic Computing: A Sample of Approaches in Minority Institutions
  - Marshall, 1980, p. 238
- Computer Enhancement of Cultural Transition
  - Gauss, 1979, p. 60
- Computer Use in Chemistry at a Minority Institution
  - Beck, 1980, p. 245
- Computing in Minority Institutions: 1976-1977
  - Marshall, Lewis, Jones, and Hamblen, 1979, p. 65
- Educational Use of Computers in Puerto Rico
  - Anger, 1980, p. 249

Pedagogy
  Computer-Assisted Test Construction via Automatic Program Generation: Using PROBGEN II to Create Individualized Exams and Problem Sets
    Collins and Duff, 1979, p. 114
  Computer Dialogs to Aid Formal Reasoning
    Arons and Bork, 1979, p. 130

Pre-College Environment
  A Computer Loan Service
    Dietz and Kemper, 1979, p. 161
  Management of Instructional Computing: Advice from Ten Pre-College Institutions
    Hunter and Hargan, 1979, p. 164
  OZNAKI and BEYOND
    Cohen, 1979, p. 170

Pre-College Instructional Materials
  Computer-Based Instruction for the Public schools: A Suitable Task for Microprocessors?
    Taylor, 1980, p. 223
  Microcomputer/Videodisc CAI -- Some Development Considerations
    Thornkildsen and Allard, 1980, p. 230
  The Non-Technical Factors in the Development of CAI
    Mocciola, 1980, p. 236

Pre-College Teacher Education
  ACM Elementary and Secondary Schools Subcommittee Progress Report
    Moursand, 1980, p. 125
  An Analysis of Computer Education Needs for K-12 Teachers
    Milner, 1979, p. 27
  A Computer Workshop for Elementary and Secondary Teachers
    Dersham and Whittle, 1980, p. 65
  Computing Competencies for School Teachers
    Taylor, Poirot, and Powell, 1980, p. 130
  Computing Competencies for School Teachers: A Preliminary Projection for All But the Teacher of Computing
    Taylor, Hamblen, Powell, and Poirot, 1979, p. 39
  Computing Competencies for School Teachers: A Preliminary Projection for the Teacher of Computing
    Taylor, Hamblen, Powell, and Poirot, 1979, p. 36
  Stages of Development in Introducing Computing to Teachers
    Dennis, 1979, p. 31

Science
  Classical Mechanics with Computer Assistance
    Davis, 1980, p. 86
  Demographic Techniques in Ecology: Computer-Enhanced Learning
    Gatz, 1980, p. 76
  Five Applications of Computers in Chemical Education
    Johnson, 1979, p. 92
  Microcomputers as Laboratory Instruments: Two Applications in Neurobiology
    Olivo, 1980, p. 81
  Quantum Chemistry Instructional Package Using Interactive FORTRAN
    Joshi and Eilers, 1979, p. 86

Social Science
  Teaching Demography Through the Computer
    Stack, 1979, p. 211
  The Use of Census Data Analysis to Enhance the Sociology Curriculum
    Benenson and Just, 1979, p. 221

Statistics
  A Computer-Based Laboratory for General Statistics
    Cunningham, 1979, p. 291
  Teaching Statistics and Using Computers
    Preskenis and Oberg, 1979, p. 297

Structured Programming
  FORTRAN 77: Impact on Introductory Courses in Programming Using FORTRAN
    Friedman, 1980, p. 103
  Shall We Teach Structured Programming to Children?
    LaFrance, 1980, p. 261
  Structured Gaming: Play and Work in High School Computer Science
    Moshell, Amann, and Baird, 1980, p. 266
  Structured Machine Language: An Introduction to Both Low- and High-Level Programming
    Hannay, 1980, p. 119
  The Use of Programming Methodology in Introductory Computer Science Courses
    Alpert, 1980, p. 96
  Using Model-Based Instruction to Teach PASCAL
    Czejdo, 1980, p. 112

Testing/Placement
  Computer-Managed Placement in Mathematics Instruction for Health Occupations Students
    Boyle and Magnant, 1980, p. 214
  Microcomputer-Assisted Study and Testing System
    Garraway, 1980, p. 200
  RIBYT -- A Database System for Formal Testing and Self-Assessment
    Fuhs, 1980, p. 205

Tools and Techniques for Instruction
  Considerations and Guidelines for Developing Basic Skills Curriculum for Use with Microcomputer Technology
    Caldwell, 1980, p. 31
  A Dynamic Process in Teaching Techniques
    Effarah, 1980, p. 25
  Hypertext: A General-Purpose Educational Computer Tool
    Ward and Bush, 1980, p. 19